1994-06

The effects of budget cuts on Army materiel command post deployment software support facilities

Jones, Mark C.

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/28144

The Effects of Budget Cuts on Army Materiel Command Post Deployment Software Support Facilities

by

Mark C. Jones
Captain, United States Army
B.S., Washington State University, 1983

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN MANAGEMENT

from the

NAVAL POSTGRADUATE SCHOOL
June 1994

REPORT DOCUMENTATION PAGE

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE June 1994	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE The Effects of Budget Cuts on Army Materiel Command Post Deployment Software Support Facilities	5. FUNDING NUMBERS
6. AUTHOR(S) Mark C. Jones	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE *A

13. ABSTRACT *(maximum 200 words)*

Increasingly, Department of Defense (DoD) weapon systems are becoming more software dependent. The future holds a ten-fold increase in the amount of on-board software in military systems. Software will provide more functionality and there will be more of it. The growth in the amount of fielded software has increased the requirements for software support services. It is estimated that more than 70% of the DoD expenditure for software is for what is commonly referred to as post deployment software support (PDSS), i.e., software maintenance of fielded system software. This thesis examines the impact of the declining defense budget and personnel reductions on Army software support activities, and the potential effect on operational systems. A secondary question was to review what is involved in PDSS support and what missions the PDSS centers perform. During this research the PDSS centers at CECOM and MICOM were examined. They provide "cradle to grave" software support. While both PDSS centers have experienced a constant growth in the number of systems they support, the number of people they have on hand has actually decreased. At the same time their budgets have not increased proportionately to their increased workload. Support for some systems has been terminated because of the cut backs. Continued budget cuts could jeopardize their ability to provide support to many systems in the future.

14. SUBJECT TERMS PDSS, Software Maintenance, Budget			15. NUMBER OF PAGES *117
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)

ABSTRACT

Increasingly, Department of Defense (DoD) weapon systems are becoming more software dependent. The future holds a ten-fold increase in the amount of on-board software in military systems. Software will provide more functionality and there will be more of it. The growth in the amount of fielded software has increased the requirements for software support services. It is estimated that more than 70% of the DoD expenditure for software is for what is commonly referred to as post deployment software support (PDSS), i.e., software maintenance of fielded system software. This thesis examines the impact of the declining defense budget and personnel reductions on Army software support activities, and the potential effect on operational systems. A secondary question was to review what is involved in PDSS support and what missions the PDSS centers perform. During this research the PDSS centers at CECOM and MICOM were examined. They provide "cradle to grave" software support. While both PDSS centers have experienced a constant growth in the number of systems they support, the number of people they have on hand has actually decreased. At the same time their budgets have not increased proportionately to their increased workload. Support for some systems has been terminated because of the cut backs. Continued budget cuts could jeopardize their ability to provide support to many systems in the future.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

ADM	Acquisition Decision Memorandum
AMC	Army Materiel Command
AMCCOM	Armament Munitions Chemical Command
ANG	Air National Guard
ATACMS	Army Tactical Missile System
ATCOM	Aviation Troop Command
C^3	Command Control Communications
CDR	Critical Design Review
CECOM	Communication Electronics Command
CINC	Commander in Chief
CSC	Computer Software Component
CSCI	Computer Software Configuration Items
CSU	Commuter Software Unit
CY	Calendar Year
DEMVAL	Demonstration Validation Phase
DoD	Department of Defense
DP	Data Processors
EMD	Engineering Manufacturing Development Phase
FCA	Functional Configuration Audit

FY	Fiscal Year
GAO	General Accounting Office
GPA	Grade Point Average
GS	General Services
GUI	Graphical User Interfaces
HSC	Health Services Command
INST	Instruction
IDD	Interface Design Document
ISC	Information Services Command
LCSEC	Life Cycle Software Engineering Centers
MICOM	Missile Command
OMA	Operations Maintenance Army
OSHA	Occupational Safety and Health Administration
PA	Procurement Funding
PCA	Physical Configuration Audit
PDSS	Post Deployment Software Support
PEO	Program Executive Officer
PM	Program Manager
R&D	Research and Development
RIF	Reduction in Force
RDEC	Research Development Engineering Center
SDD	Software Design Document

SED	Software Engineering Directorate
SSR	Software Specification Review
SSS	System/Segment Specification
STP	Software Test Plan
TBM	Tactical Ballistic Missile
USD(A)	Under Secretary of Defense for Acquisition

I. INTRODUCTION

A. GENERAL

Increasingly, Department of Defense (DoD) weapon systems are becoming more software dependent. The challenge facing DoD is similar to the challenge facing the civilian sector. Both are experiencing an incredible growth in the amount of software that they support. In 1977, UNIX (version 6) kernel had only 11,000 lines of code. One individual could read and understand all of the code by studying it for several weeks. In 1990 a version of the same item had expanded to 450,000 lines of code. The code now covers 7,500 pages. The printed program is so heavy (75 pounds) that it is against Occupational Safety Health Administration (OSHA) regulations for one person to lift it. In 1977 the entire UNIX system had between 200,000 and 400,000 lines of code. In 1990 that same system had expanded to 1,900,000 lines of code. DoD is experiencing a similar software growth pattern. (ARES, 1991)

The General Accounting Office (GAO) estimates that currently DoD spends between $24 billion and $32 billion annually on software (GAO, 1992, p. 2-3). We cannot get any better estimate (an $8 billion range) because DoD has not had a means to track software costs (GAO, 1992, p. 2-3). In the past software has not been tracked as a separate, discrete item. The estimates listed above were obtained by the GAO from external sources such as trade associations and defense media. However, this lack of cost data has been recognized and steps to correct it have been initiated. A change to Defense

Instruction 5000.2 now requires that any weapon system started after February 23, 1991, should identify and report software costs separate from other costs (DoD Inst 5000.2, 1991). It is also estimated that more than 70% of DoD's expenditure for software is for Post Deployment Software Support (PDSS), commonly referred to as maintenance of fielded software (ARES, 1991, p. 9-12).

In the last 25 years the DoD requirements for software have skyrocketed. Software costs for weapon systems have been dramatically rising. It is expected they will continue to rise at a disproportionately greater rate than other system costs.

Software will be providing more functionality and we will be getting more of it. This increase in the amount of fielded software has increased the requirements for software support services. Currently there is a shortage of qualified software personnel. The demand for software support personnel is increasing at a rate of 12% per year, while the actual number of software personnel is increasing at a rate of only 4% year. (MCCR, 1990, ch. 7, p. 2)

The software programs being delivered are much larger in size, and therefore more complex. This makes them more difficult to maintain. Unfortunately, there has often been a tendency, when programs dollars are reduced, to cut out many of the items that are needed to support the software once fielded. These include proper documentation and software support tools.

With the current downsizing of the military and a decreasing DoD budget, the military support infrastructure is also being significantly reduced. This poses a dilemma in the Government software community. At a time when greater amounts of software are

being supported, the number of personnel to support it is being reduced at an alarming rate. This downsizing is creating significant challenges for the software support community because software maintenance is an extremely manpower-intensive activity requiring specific program knowledge to be productive. Many programmers who are familiar with existing programs are being lost. Their experience and knowledge, perhaps the most important factor in software productivity and quality, are assets that cannot be easily replaced. (ARES, 1991, p. 3-1, 3-14)

The seriousness and magnitude of the PDSS problem must be communicated to senior Army leadership. The risks and potential disastrous consequences to our weapons, command, control, and communication systems must be addressed.

Software plays a critical role in today's modern weapon systems. It must be maintained. Thus, PDSS should be thought of in the same light as we think of any of our other production infrastructures.

We in the Army have reached a point where software support for some systems is being terminated, brought about by budget and personnel cuts. Caution needs to be taken in cutting the infrastructure that provides software support for the software-intensive weapon systems our soldiers in the field will use in future conflicts. If the software support for a system is terminated, then removal of the system from the fielded inventory must be given serious consideration. Commanders would not think of keeping weapon systems for which logistics support had been terminated. (ARES, 1991, p. 3-1, 3-14)

B. AREA OF RESEARCH AND OBJECTIVES

This thesis examines the PDSS of Army weapon systems and the impact of personnel and budgetary cutbacks on the ability of the PDSS centers to perform their assigned missions.

C. RESEARCH QUESTIONS

The following questions pertain to this research effort.

1. Primary Question

What is the impact of the declining budget and personnel reductions on Army software support activities, and what are the potential effects on operational weapon systems?

2. Subsidiary Questions

a. What is the function of the PDSS activity?

b. What is involved in PDSS?

c. What was the role of the PDSS activities in supporting Desert Shield/Storm?

d. What alternatives are available to the PDSS activities in implementing personnel and budget cuts?

D. SCOPE

This thesis investigates the impact of budgetary and personnel reductions on the Army Materiel Command (AMC) PDSS activities. It does not address the PDSS centers

located in either the Health Services Command (HSC) or the Information Services Command (ISC). Because of time and funding constraints, this thesis will only examine the AMC PDSS centers located in the Communication Electronics Command (CECOM) and the Missile Command (MICOM).

E. METHODOLOGY

1. Literature Search

A literature search was conducted in order to gain a background on software maintenance. Department of Defense (DoD) literature was searched in order to gain knowledge on the DoD software life cycle.

2. Survey

A written questionnaire was prepared. (See appendix) This questionnaire was sent to all individuals who were going to be interviewed so that they would be adequately prepared for the telephone interviews and site visits.

3. Interview

All individuals at the PDSS centers were first interviewed by telephone, followed by an on-site visit.

F. LIMITATIONS

As previously mentioned, because of time constraints and limited funding this thesis deals only with two of the AMC PDSS centers. The PDSS center located in Aviation Troop Command (ATCOM) only performs software management, and is not a code

producing facility. The PDSS center located in Armament Munitions Chemical Command (AMCCOM) primarily does PDSS and provides very little software life cycle support. The research's focus is on centers that perform support on weapon and Command, Control, Communication (C^3) systems software. No attempt was made to look at activities providing support for automated information systems, Health Services systems, or Logistics systems.

G. ORGANIZATION

Chapter II establishes background for the software maintenance challenge in general. It is based on a literature review of the field. Software maintenance is discussed from a generic point of view, rather than drawn from a specific DoD point of view.

Chapter III reviews current literature on how software maintenance is performed, again not specifically in the DoD context.

Chapter IV reviews the DoD acquisition life cycle, and discusses how software development relates to this process. Material for this chapter was obtained from the DoD Instruction 5000 series as well as the Air Force Acquisition Model program.

Chapter V discusses the methodology and major areas of research that will be addressed in Chapter VI.

Chapter VI summarizes all of the telephonic interview data that was gathered. It also discusses any information gathered during on-site visits.

Chapter VII presents the conclusions and recommendations derived from the data presented in Chapter VI.

II. THE SOFTWARE MAINTENANCE CHALLENGE

The first section of this chapter examines the software maintenance challenge. The following section presents what current literature says about software maintenance. Individuals who are already familiar with software maintenance may skip this chapter.

A. WHAT IS THE CHALLENGE?

Many people understand that as hardware gets older, it starts to deteriorate, is hard to change, requires preventive maintenance, and has high maintenance costs. It is now common knowledge among software professionals that software maintenance is important, difficult, and expensive. Unfortunately, all too many senior managers who control resources appear to have the erroneous perception that exactly the opposite is true for software. They believe it improves with age, is easy to change, does not require preventive maintenance, and is less expensive than the original cost.

In the book Software Maintenance (Martin, 1983) the authors define software maintenance as.

> ...changes that have been made to a computer program after they have been delivered to the customer or user. (Martin, 1983, p. 3)

It is obvious to most people when maintenance needs to be performed on a piece of hardware equipment. Usually the greatest resources are used for maintenance that is performed when the hardware is actually broken. In addition, as in the case of an

automobile, we also perform routine preventive maintenance, such as inspections, fluid changes, etc.

The same reasoning or association with the term maintenance does not directly carry over to software maintenance. As is shown in Figure 1, only 20% of the maintenance effort is conducted on broken software (i.e., to fix bugs). The remaining 80% of changes primarily involves fine tuning and user-demanded enhancements. In keeping with our previous definition of maintenance, there are a number of reasons why maintenance is conducted on software. (Lientz, 1980, p. 67-96)

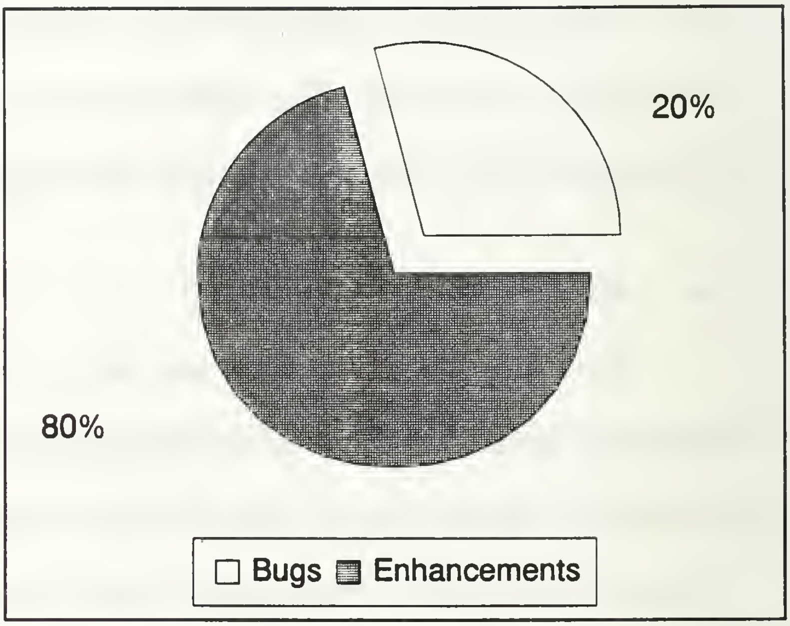


Figure 1 Bugs as Percentage of Maintenance Actions (Martin, 1983, p. 4)

Figure 2 illustrates the three types of maintenance that are discussed in a Lientz and Swanson survey: perfective, adaptive, and corrective. They define perfective maintenance to be that maintenance performed to enhance performance and improve cost-effectiveness, improve processing, improve efficiency, and improve maintainability. They define adaptive maintenance to be that maintenance performed to adapt software to changes in the data requirements or processing environments. They define corrective maintenance as any maintenance performed to identify and correct software failures,

performance failures, and implementation failures. This chart demonstrates the fact that the perfective and adaptive maintenance account for 80% of software maintenance. Of these two, perfective maintenance accounts for 55% of the software maintenance performed. (Lientz, 1980, p. 67-96)

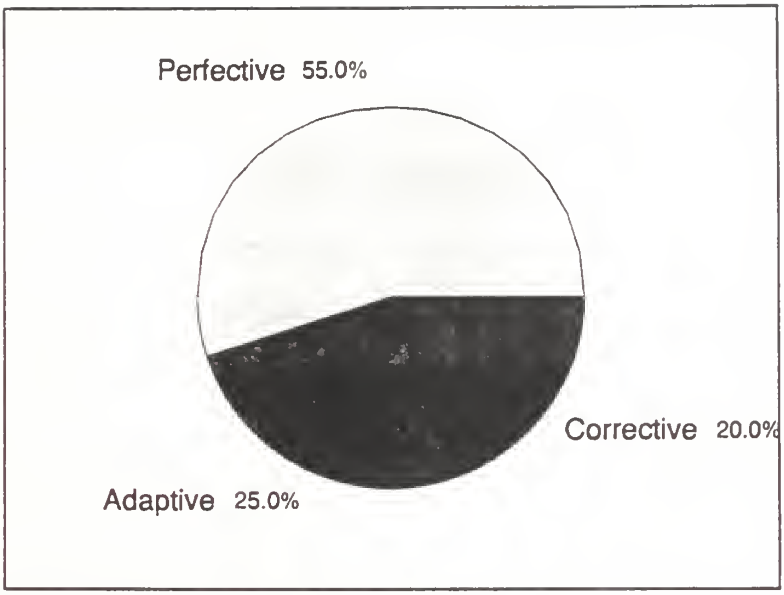


Figure 2 Breakdown of Types of Maintenance (Martin, 1983, p. 29)

Figure 3 breaks perfective maintenance down into its major components. Of the perfective maintenance performed by software maintenance organizations, over 75% is to install user-requested enhancements. (Lientz, 1980, p. 67-96)

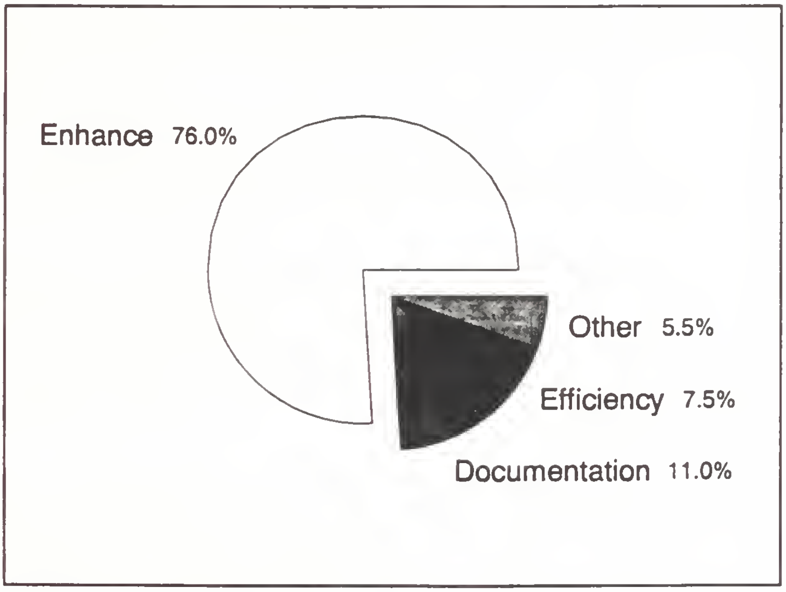


Figure 3 Breakdown of Perfective Maintenance (Martin, 1983)

Figure 4 examines user-requested enhancements by itself. Of the user requested enhancements, 70% of those are requested by the user to gain more information. (Lientz, 1980, p. 67-96)

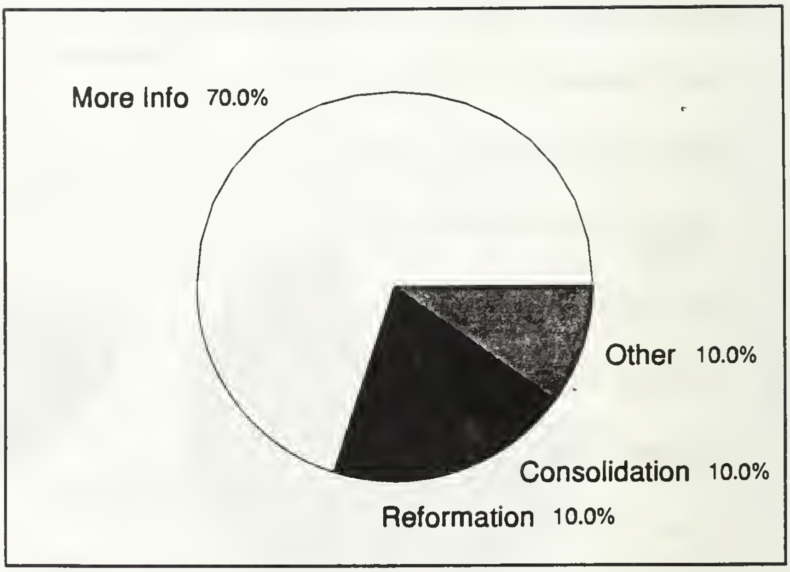


Figure 4 Breakdown of User-Requested Enhancements (Martin, 1983)

The fact is, when software maintenance is conducted, it involves more than just correcting defects or making design enhancements; it also involves making enhancements that change the software functionality and how the program itself behaves.

An article in the Wall Street Journal stated that "oil and software are the two principal obstacles to economic progress." (Buckley, 1980, p. 1,18) More and more software users are becoming comfortable with computers. They are beginning to expect the easy interfaces characterized by graphical user interfaces (GUIs) such as Windows or Apple Macintosh. It should be expected that software users become dissatisfied and impatient because of system bugs and failures. However, they are also becoming dissatisfied with poor documentation, inadequate training, and the inability of programmers to meet their changing requirements in a timely manner.

Much of the software maintenance problem comes from not designing software for future maintenance in the first place during initial development. The hidden cost is often

much higher than just acquiring difficult-to-maintain software. Such software can become fragile to the point where managers are reluctant to make changes. (Pizzarello, 1984, p. 1-17)

Unlike hardware maintenance, software maintenance tends to deteriorate the structure of a program. Maintenance changes often end up making the program structure more complex and brittle. The documentation quality also tends to deteriorate. This combination, and other factors, make it more difficult to maintain the software the next time maintenance is required. (Pizzarello, 1984, p. 1-17)

Some managers erroneously perceive that conducting maintenance is a lot like conducting original programming, perhaps even easier. This is a myth. Martin brings out that new errors are invariably introduced during the maintenance process. As these errors are introduced, more and more time is spent correcting these secondary problems rather than fixing the structure that caused the original problem. (Pizzarello, 1984, p. 1-17)

The software maintenance challenge is best summarized by F. Brooks.

> ...systems program building is an entropy-decreasing process, hence inherently metastable. Program maintenance is an entropy-increasing process, and even its most skillful execution only delays the subsidence of the system into unfixable obsolescence. (Brooks, 1975, p. 41-50)

It comes down to the fact that if you want the software to be easy to maintain, one should begin when the system is originally conceived. There are many actions that can be performed to minimize maintenance costs. Some of these are:

- Conduct the system design and programming from the start with the intention of minimizing maintenance.

- Adopt management practices that are sound for maintainability.

- Minimize the complexity of system and program structures.

- Select and enforce structured techniques that lead to ease of maintenance.

- Reduce the need for corrective maintenance by improving software reliability/quality.

- Reduce the need for change maintenance by planning for and controlling user enhancements and by using table-driven (i.e., parameterized) programs.

- Anticipate possible migration to new technologies or software, and plan to minimize the need for program rewriting.

- Ensure that the documentation and structure diagramming will be entirely clear for future maintainers. (Martin, 1983, p. 12-14)

B. WHAT IS MAINTENANCE?

The following paragraphs discuss some background on software maintenance. After the background discussions, methods of minimizing the costs of performing corrective, adaptive, enhancement, and supportive maintenance will be discussed.

1. Background

Two individuals who have done a great deal of work on classifying maintenance are John Reutter and E. Burton Swanson. Swanson's work says that you can break maintenance down into three basic categories. These categories are summarized in Figure 5.

Failures can be attributed to errors in the programs. Some of these errors include invalid output results, missing data edit checks, performance inefficiencies, and programming standard violations. (Swanson, 1976, p. 492-497)

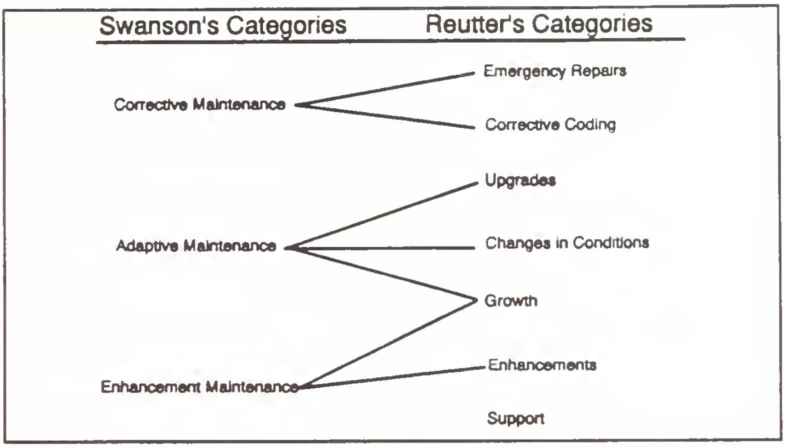


Figure 5 Swanson and Reutter's software maintenance categories (Martin, 1983, p. 22)

Environmental changes are a common occurrence in organizations over a period of time. Two types of environmental changes can be expected. The first type is a change in the data environment, which includes changes in laws or regulations. The second type that can be expected is a change in the processing environment, such as the installation of new hardware or a new operating system. (Swanson, 1976, p. 492-497)

Users and maintainers are also a cause of maintenance. Changes are requested to improve a software program's operating efficiency, to add new features, to change existing functions, or to improve maintainability. (Swanson, 1976, p. 492-497)

While Swanson and Reutter both use similar classification schemes to identify the basic causes of maintenance, there is one important difference. Reutter puts support into a separate category, whereas Swanson includes support in all three of his categories. Reutter separates support from the remaining categories to emphasize the importance of maintenance. He also does this to emphasize the importance of communication between

the users and the maintainers, as well as the importance of planning for support. (Martin, 1983, p. 20)

Software Maintenance summarizes the surveys conducted by Lientz, Chrysler, and Chapin. Surveys conducted as part of their investigations concluded that the following were major characteristics that effect the maintenance effort.

- System size.
- System age.
- Number of input/output items.
- Application type.
- Programming language.
- Structuredness (Martin, 1983, p. 25-26).

2. Maintenance Overview

Reexamination of Figure 1 reminds us that the greatest contributors to software maintenance are enhancements performed at the request of the user. Ideally we need to examine the sources of software maintenance, and reduce the amount of maintenance that must be conducted.

3. Corrective maintenance

Ideally, maintainers would like to eliminate the need for corrective maintenance all together by ensuring that software is as reliable as possible when it is

initially designed. Some methods that can be used to minimize the need for conducting corrective maintenance are listed below. (Martin. 1983, p. 29-30)

- Data-base management systems.
- Application development systems.
- Program generators.
- Very high-level (fourth-generation) languages.
- Application packages.
- Structured techniques.
- Parameterized programs.
- Defensive programming.
- Maintenance audits (Martin, 1983, p. 29-30).

The first four methods on the list above tend to produce more reliable code because a significant portion of the code is generated automatically (i.e., the amount of manual coding is reduced). In addition, Martin submits that application packages tend to have a higher reliability than single-user custom-coded systems. This is because application packages have multiple users, and more users tend to find more errors.

It is common knowledge that a program that has been developed using structured techniques is easier to understand and to test. This results because the control structure has been standardized and the number of program paths has been reduced by structuring restrictions.

Defensive programming introduces self-checking capabilities into a program. Self-checking works by checking for off-nominal situations, providing audit trails, and flagging unsafe programming practices. The last item in Martin's list, maintenance audits, helps identify quality deficiencies before they cause maintenance problems. (Martin, 1983, p. 29-30)

4. Adaptive Maintenance

While attempting to minimize maintenance, we must realize that adaptive maintenance cannot be completely eliminated. Hardware and software "state of the art" change rapidly; it is inevitable that some of this improving technology will be introduced. Martin, in his book on software maintenance, cites the following examples of how to control adaptive maintenance:

> ...using configuration management to plan for computer hardware and operating system changes can reduce the need for some adaptive maintenance work. Also, isolating system-dependent features into special program modules can limit the portion of a program that must be modified to accommodate configuration changes. Finally, using internal program tables/arrays, external files, and packaged routines to handle special processing (e.g., Government regulation) can make programs easier to modify when adaptive changes are necessary. (Martin, 1983, p. 30)

5. Enhancement Maintenance

Enhancement maintenance can also be reduced by using many of the techniques mentioned under corrective and adaptive maintenance. In addition, the use of data base management systems, code generators, and commercial application packages can reduce some of the maintenance required. An advantage to using these is that it often

gives the user an opportunity to directly perform some of the enhancements themselves. (Gibson, 1986, p. 17-36)

Another method to improve the productivity of enhancement maintenance is to build a prototype system. A prototype system built early in the software development cycle allows users to work with the system and further refine their requirements. This will decrease the number of future maintenance requirements. (Gibson, 1986, p. 17-36)

6. Supportive Maintenance

The final type of maintenance is supportive maintenance. Supportive maintenance can be reduced by implementing a number of things. The first method is to ensure that users have up-to-date documentation. If the users documentation is up to date, the maintainers will receive fewer questions. A second method is to have on-line user documentation. This on-line documentation is the most convenient and allows users to get answers to their questions when they have the problem. A third method for reducing support maintenance is to ensure that the users have adequate training. The premise behind the items discussed above is that the better the system users understand the software, the less support they will require from the systems and programming staffs. The final method for reducing support is to have a separate staff devoted solely to software maintenance. A separate maintenance staff allows the maintainers to track problems better and to remain familiar with the code design, punctuation, logic, and documentation. A problem that can develop from this approach is that the maintainers can be thought of as second class citizens. The experienced programmers develop programs while the inexperienced programmers maintain them. (Martin, 1983, p. 32-38)

III. PERFORMING SOFTWARE MAINTENANCE

This Chapter discusses the issue of program maintenance, the debugging process, and managing maintenance. Readers who are familiar with this process may skip this chapter.

A. PROGRAM MAINTENANCE

1. Introduction

Many of the problems involved in software maintenance (i.e., personnel reduction, funding cut-backs, etc.) are caused in part by the mistaken belief that software maintenance is substantially different and generally easier than program development. It is now common knowledge among software professionals that software maintenance is complex and resource-demanding. Unfortunately, all too many senior managers who control resources have unrealistic perceptions about software maintenance. The misperception that software maintenance is easier then program development leads to ideas that it requires less experienced personnel, less sophisticated tools, and less management control. (Martin, 1983, p. 361)

Usually, just the opposite is true. Several factors need to be considered. First, the software maintainer is required to understand programs that were written by somebody else. This is often a very demanding task. Next, they must make program modifications without jeopardizing the existing software's correctness or integrity. Often there is an inadequate schedule which adds increasing pressure and forces trade-offs.

Much of the maintenance is performed on older legacy systems, many of which are unstructured and brittle programs with poor documentation. This often makes them difficult to maintain. There is also a scarcity of modern programming tools available in many software maintenance organizations. (Pizzarello, 1984, p. 213-229)

Software maintenance involves a variety of tasks. Some of these tasks include:

- Reviewing program requirements and specifications.
- Interviewing end users and developers.
- Examining programs and documentation.
- Determining the cause of program errors (20%) and where program changes should be made (80%).
- Evaluating the possible side effects resulting from a program change.
- Coding program changes.
- Revalidating programs.
- Updating program documentation and libraries (Martin, 1983, p. 361).

Many problems arise from software maintenance. Just as errors (bugs) are introduced during new system development, some bugs will result from the software maintenance effort. It only takes a couple of haphazard software patches to make a program a maintenance mess and lead to the early obsolescence of that software. Other problems that frequently confront software maintainers are listed below: (Pizzarello, 1984, p. 213-229)

- Poor quality of the original program.
- Inadequate documentation.
- Limited tools.
- High learning curve due to the increasing size and complexity of new software systems (Martin, 1983, p. 362-363).

Regardless of the problems and risks that are created by performing software maintenance, the need to do it will not go away. A reasonable approach to mitigate these problems is to improve the maintenance process itself. These are methods and actions that can improve the process; some are listed below.

- Provide a program modification procedure that uses structured techniques.
- Encourage flow of communication among the end users, maintainers, and developers.
- Establish and enforce programming and documentation standards.
- Improve documentation for existing programs.
- Perform maintenance audits to check the correctness and quality of maintenance work.
- Increase end-user involvement and responsibility in the maintenance process.
- Batch maintenance requests rather than make program modifications in a piecemeal fashion.
- Emphasize careful and thorough retesting and revalidation when a program is modified.
- Provide continuous training for maintainers in new software technologies and to improve their knowledge of the application area. (Martin, 1983, p. 362-363).

2. The Software Maintenance Process

Typically, the software maintenance is broken into a three-step process. The steps involved will be discussed in the following sections and include:

- Understand the existing program.
- Modify the existing program.
- Revalidate a modified program (Martin, 1983, P. 363).

a. Understanding the Program

There are three steps involved in understanding a program: gain a top-down understanding, improve documentation, and participate in the development process. (Martin, 1983, p. 363)

Understanding the program involves acquiring an understanding of the functional objective of the program, its internal structure, and its operating requirements. Failure to do this could result in the maintainer inadvertently introducing new errors into the program. (Martin, 1983, p. 363)

Developing an understanding of the program is actually a three-step process. The first step is for the maintainers to be given a chance to understand the program before they modify it. Gaining this familiarity implies that the maintainer will gain an understanding of the program's overall function, its basic components, its stability, and its ability to be easily and correctly modified. James Martin explains the methodology behind the Top-Down understanding approach.

First become familiar with the overall program purpose and the overall flow or control from component to component. Identify the basic data structures as well as the processing components of the program. If this program is part of a system of programs, understand its functions within this system.

Then become immersed in the details of how the program works by identifying what each component does and how it is implemented in the code. A component may be a data file, a program subroutine, a program module, or simply a piece of program code that can be logically grouped together.

As you learn about the program, document what you learn. (Martin, 1983, p. 364)

In order to perform software maintenance on a piece of software, the most current source code listing needs to be available. Many times these listing are not available to the programmers. This complicates software maintenance and makes it more difficult.

There are automated tools available to help improve a program's readability. This helps improve the maintainers chance of successfully modifying a program.

Freedman suggests an approach called "fix and improve." The philosophy behind this method is not only to make a program modification or change, but also to improve the future maintainability of the program. For instance, if the high-level program documentation is incomplete or inaccurate, the maintainers should attempt to recreate it and to improve it. (Freedman, 1980, p. 57-59)

The final step in understanding a software program is that the maintainers should be involved in the design process if at all possible. This will give the maintainers a chance to become familiar with the program prior to accepting responsibility for its

maintenance. The maintainers would thus be involved in the design, coding, and testing phases of the software developmental life cycle.

b. Modifying the Existing Program

Modifying the program involves three steps: design the change, alter the code, and minimize the side effects of the program change.

Swanson lists two types of program changes: correcting a program error and performing a program enhancement. If the maintainers are going to correct an error, they must first find the cause or causes of the error. They then determine what program logic must be changed to correct the error. Adding a program enhancement requires the development of new code. Once the new code is developed, the maintainers must then figure out how to integrate the new modification into the existing program. (Swanson, 1976, p. 492-497)

Once the maintainers have completed these steps, they can actually alter the code. There are two objectives when altering a program's code: correctly and efficiently code the change, and most important for the user eliminate any unwanted side effects from the change.

Freedman divides the side effects from performing software maintenance changes into four categories. These categories are code errors, data errors, documentation errors, and miscellaneous errors. (Freedman, 1980, p. 57-59)

The first two categories of errors mainly affect the program itself. These two errors are referred to as "ripple effect" errors, because they create additional errors in the program. The last two categories of errors have a much broader impact. They

have an adverse impact not only on the program itself, but on how the program interacts with other programs as well. Errors in the documentation of a program may create problems because program documentation tells other program designers how to make their program interact with the the original program.

c. *Revalidating the Program*

Once a program is modified, then the entire program needs to be revalidated. This ensures that not only have the maintainers used the correct logic when they made the change, but it also validates that the unmodified portions of the software still work.

Data indicates that if a maintenance action involves fewer than 10 lines of source code, there is less than a 50% chance of programmers successfully making that change without introducing additional errors. If the maintenance action involves 50 statements or more, the probability of successfully making a change is reduced to only 20% (Boehm, 1973, p. 48-59). The revalidation process closely resembles the original program validation effort. The maintainers use the same or similar test cases, as well as the same or similar test data.

When a program is revalidated, three steps are followed. System-level testing involves testing the entire program for failures. Once the change has passed the system-level test, then the programmers test the unmodified portions of the program by conducting regression tests. Regression testing involves running the program through an entire bank of tests to ensure that other areas of the software program still work correctly.

The maintainers directly test the modified portions of the program as the last step in the process. (Martin. 1983, p. 376-377)

B. THE DEBUGGING PROCESS

As mentioned, the chances of successfully completing a maintenance action without any errors are very small. This is where the debugging process comes in. Two basic methods often used are individual debugging or group debugging, or a combination of both.

Based on research data, the best results achieved during the debugging process are achieved using the following method. The first step involves two individuals who work independently attempting to locate errors using a computer-based testing approach. During this step in the testing process, some desktop testing is conducted. When the individuals complete their individual testing, they pool all of the errors that they have found and conduct a program walk-through using the error evidence that each has gathered separately. During the entire process, automatic debugging tools are used to aid the process. (Schneiderman, 1980, p. 129-130)

In order for maintainers to effectively debug a program, the following list of items should be prepared:

- The source code listing.
- A detailed program specification.
- A program flow chart.
- An output listing.

- A trace of statements executed and the variable values.
- Access to a terminal for program execution.
- Clues to types or location of error (Schneiderman, 1980, p. 112-113).

Debugging is a very difficult and time-consuming process. Martin provides some general guidelines to use when debugging:

- Not using a random approach to debugging.
- Isolating one error at a time.
- Embedding debugging code into the program to make program errors easy to locate.
- Carefully studying the actual program output, and comparing it to samples of the expected output.
- Focusing attention on data handled by the program rather than solely on the program's processing logic.
- Using the most powerful debugging tools available, as well as a variety of debugging methods to avoid becoming locked into considering only one possibility too prematurely.
- Keeping a record of errors detected and corrected, as well as noting where the errors occurred in the program.
- Keeping a record of the types of errors that are found, since this information will be used to predict where future errors will occur.
- Measuring program complexity.
- Training maintainers in debugging techniques using training programs involving artificially seeded errors. Immediate feedback on all seeded errors is provided showing the maintainers what they missed. (Martin, 1983, p. 391)

C. MANAGING MAINTENANCE

With regards to the software life cycle, most management attention has been paid to the development portion; most managers would prefer that the maintenance portion of the software life cycle just go away. Good software engineering practices can be applied to the software maintenance phase of the life cycle as well as the initial development portion of the life cycle. As is the case during software development, management problems often outweigh technical problems during the maintenance phase.

There is no technological change or easy fix that will get us out of the ever-growing software maintenance challenge. The project management techniques that are being applied in program development should now be applied to software maintenance. These include instituting a change control procedure, creating a separate maintenance staff, using a scheduled maintenance approach to software changes, and creating programming teams. Each of these is briefly discussed.

1. Change Control Procedures

The biggest management problem that maintenance organizations face is trying to control program change. Some methods that can be used to control the change process include creating a change review board, using a user charge-back system, and instituting a program quality control audit step. These are all methods that control the costs and risks associated with modifying programs. (Osborne, 1983, p. 38-40)

Requests for program changes come from many places. They come from the user's community, from the program development group, from the maintenance community, from the operations community, and from management.

Regardless of where the maintenance action originates, no program modification should be made until the changes have been carefully considered. Even simple changes can have major consequences to a program. Careful consideration needs to be given to each change. With different groups submitting changes, it is possible that conflicting changes will be forwarded to the maintainers. In addition, a community may ask for a change that is incompatible with the life cycle goals of the program. An example of such a change would include one user community requesting a software enhancement that suboptimizes the program for the remaining users. (Osborne, 1983, p. 38-40)

Some questions should be answered when changes are considered. Is the maintenance objective of the program to preserve a single version of the program for all users, or is to allow different versions to support different user groups? Is there a plan to replace this program in the near future? Will this change alter the original scope and purpose of the program? (Osborne, 1983, p. 38-40)

Effective change control procedures have three primary purposes. The first is to study how the program changes over time so that future support needs can be planned. A second is to provide follow-up procedures to every user request, because it is important to report back to the requester with the planned change implementation schedule or the reason for rejecting the request. The third is to ensure that changes are planned and scheduled so that maintenance tasks can be planned and scheduled. (Martin, 1983, p. 414-418)

Martin suggests the best method of controlling change is by using two concepts. The first method is to appoint a change review board whose job it is to review all change requests. The second method is to implement a charge-back system. Under this system the users pay for all changes that they want to implement. This may create problems. Those organizations that can afford it will get all of the changes; those sections unable to fund changes will get no changes to their programs, even though their requirements may be just as important. (Martin, 1983, p. 414-418)

2. Separate Maintenance Staffs

Another method of dealing with maintenance management problems (e.g., by increasing programmer productivity, controlling the maintenance effort, and controlling costs associated with performing program maintenance) is to create a separate program maintenance staff. (Martin, 1983, p. 421-422)

The advantage of creating a separate maintenance organization is that it is possible to raise the morale of both the maintenance and development staff. Normally, only larger organizations can afford the cost of a separate maintenance staff. As mentioned earlier, when creating a separate maintenance organization care must be taken to ensure that not just inexperienced programmers are assigned to the maintenance staff.

3. Scheduled Maintenance

Scheduled maintenance uses the new release concept for systems software. When program changes are consolidated, managers are given more time to plan for

efficient program modifications. In addition, the users know that their requests will not be acted on immediately. This forces them to give careful consideration to which changes they really need. When the changes for any single program are batched together, the impact of the changes can be more thoroughly evaluated. When management knows which applications will be maintained during which months of the year, they can plan more effectively and prioritize the maintenance projects. (Martin, 1983, p. 421-422)

4. Programming Teams

The management problem can also be reduced with the creation of programming teams. These teams provide the same benefits to software maintenance as they provide to new software development, i.e., improved program quality, increased programmer productivity, and job enrichment.

Osborne provides three popular team programming concepts. These concepts are peer reviews, chief programmer team, and the walk-through.

a. Peer Reviews

The peer review method is the oldest and simplest programming method. The premise behind it is that everyone connected with a program's development will work together to build the best program possible. This method emphasizes an open, democratic environment in which personal worth is separate from errors in one's work. The quality is assured in this method when the programmers review and critique each other's work. (Osborne, 1983, p. 38-40)

Sections of code developed by one programmer are given to another programmer, who then identifies what he considers to be errors. It is important, though, to ensure that everybody involved in the process knows that this is not an evaluation of the programmer's capabilities or performance. Instead, the process provides an analysis and evaluation of the code. If at all possible, the project manager should not be involved in the peer review. (Osborne, 1983, p. 38-40)

b. Chief Programmer Team

The chief programmer team is completely opposite the democratic structure of the peer review programming method. Under the chief programmer team concept there is a strict organizational structure in which discipline, clear leadership, and functional separation are stressed. (Osborne, 1983, p. 38-40)

This programming method is based on the concept that an experienced programmer who is supported by a team of programmers can produce a quality product much faster than the groups of programmers organized under traditional methods of organization. Under this concept, the chief programmer is responsible for the overall design, development, review, and evaluation of the work performed by members of his team. This could include the establishment and enforcement of rules regarding programming style, control, and integrity of the program.

In order for this to work, the chief programmer must be the focal point of the programming effort. He must be aware of and familiar with all of the work performed by the team. This method places a large administrative and technical burden on the chief programmer. He or she must have strong leadership abilities, technical

capabilities, and the ability to delegate work and responsibility. An important part of the chief programmer concept is that other members of the team support the chief programmer - e.g., prepare test data, run tests, maintain the program library, document programs. (Osborne, 1983, p. 38-40)

c. *Walk-Throughs*

The structured walk-through concept involves a set of formal rules for reviewing a program's development progress. One advantage is that it may be used during any phase of the program life cycle. It may be used to determine schedule breakdowns, to identify development problems and system errors, or to provide a constructive learning environment. The basic rules for this approach are:

- A walk-through is not used to review an individual's work performance; it is used to review program quality and project progress.
- Each person attending a walk-through must function as a reviewee, reviewer, or recording secretary.
- During the walk-through, problems may be identified, but not corrected. (Osborne, 1983, p. 38-40)

D. CONCLUSION

This Chapter has presented information obtained from literature on performing software maintenance. In particular it has discussed program maintenance, debugging processes and managing maintenance. The following chapter will discuss the acquisition and software life cycles.

IV. THE SOFTWARE LIFE CYCLE

All of the PDSS centers that were examined mentioned the same thing, their software support mission is a cradle-to-grave responsibility. Many people are under the mistaken belief that the PDSS centers are involved strictly in PDSS. This is not true. The PDSS centers provide much of the matrix support required by program managers during the early developmental phases of a weapon system's life cycle. In addition, the eventual efficiency of the maintenance of the system software may be significantly affected by decisions and actions taken at the initiation of the software development effort. The following paragraphs discuss the acquisition and software life cycles.

A. THE ACQUISITION LIFE CYCLE

DoD acquisition programs are managed through a series of progressive acquisition phases, with major milestone decision points. DoD Instruction 5000.2 defines five major milestones and phases of the DoD acquisition process. These milestones and phases are the basis for all weapon system management in DoD.

1. Pre-Concept Exploration and Definition

The major objective of the Pre-Concept Exploration and Definition phase is to take the validated mission need statement and decide if that mission need statement warrants the initiation of further studies. If further studies are warranted, then the study group will decide how many alternate ideas will be investigated. (AFAM, 1993)

The program must complete several tasks before it can leave this phase. These tasks include validating the requirement, developing an acquisition strategy, and awarding the concept evaluation contract or contracts. (AFAM, 1993)

Once the Defense Acquisition Board (DAB) agrees that a requirement is valid, then the Under Secretary of Defense for Acquisition (USD(A)) issues an Acquisition Decision Memorandum (ADM). The ADM issued by the USD(A) identifies several key items. These items include which service and office has responsibility for the program. The ADM will determine how many alternatives will be examined. It will also identify the lead organization for the study, the funding limits, the source of funding, and the Milestone I exit criteria.

2. Concept Exploration & Definition-Phase 0

During the Concept Exploration and Definition phase, threat-based mission feasibility studies are done. At this early stage, the studies will not address any specific hardware solution.

The purpose behind these studies is to decide which concepts or technologies can satisfy the mission need statement. These studies will also identify any high-risk areas, as well as a management approach. The program acquisition strategy, program initial cost objectives, program schedule, and system performance are identified for all of the most viable alternatives. The conclusion of this phase is Milestone I Concept Demonstration Approval.

3. Demonstration & Validation-Phase I

The Demonstration and Validation phase (DEMVAL) serves to validate the critical technologies and processes that will be used for system development. During this phase the contractor will prove that he understands the processes involved in the program. The characteristics and capabilities of the system are more clearly defined. The program office also identifies which design approach they prefer. This phase is concluded with the Milestone II Development Approval. (AFAM, 1993)

4. Engineering & Manufacturing Development-Phase II

Significant portions of the systems developmental resources are consumed during the Engineering and Manufacturing Development phase (EMD). The subsystems developed during DEMVAL are integrated into a full-up system during EMD. The most promising system developed during EMD is advanced into a stable, producible, and cost-effective system design.

The exit point for this phase is the Milestone III production approval decision. This milestone gives approval for the system to proceed into either low-rate or full-scale production.

5. Production & Deployment-Phase III

The objective of the Production and Deployment phase is to establish a stable, efficient production and support base. It is during this phase that an operational capability that meets the user's needs are achieved. To confirm and monitor performance and system quality, follow-on operational testing is conducted. Another method that is used

to monitor quality is to perform production verification testing. Verification is conducted to ensure that deficiencies are being corrected. During this phase any improvements that were not incorporated into the original design can be scheduled into future production lots.

6. Operations & Support-Phase IV

The Operations and Support Phase of the life cycle begins when the users declare that the system is operationally capable. This phase continues until the system is retired and leaves the inventory. The objectives of this phase are to ensure that the fielded system continues to provide the capabilities necessary to meet the documented mission need. In addition, if the system was fielded with any performance deficiencies these may be identified and system upgrades to fix these may be introduced during this phase. (AFAM, 1993)

B. SOFTWARE LIFE CYCLE

The development of a new weapon system requires numerous skills. It requires the integration of technical, administrative, and management disciplines. Integrating these disciplines creates a cohesive, well planned, and rigorously controlled development process. As computers become an ever more critical part of today's modern weapons systems, software also takes on a more critical role.

The software development process must begin early in the weapon's system development process. It begins during the early stages of the system engineering process. Once the system engineering process commences, the system is broken into major

components. This includes deciding what portion of the mission will be performed by hardware and what portion will be performed by software. Once these requirements have been defined, the software development process begins.

1. Software Development Process

DOD-STD-2167A breaks the software development process down into an eight-step development cycle. The discussion in this DoD standard revolves around what is commonly referred to in the software world as the "waterfall model." Figure 6 graphically depicts this process.

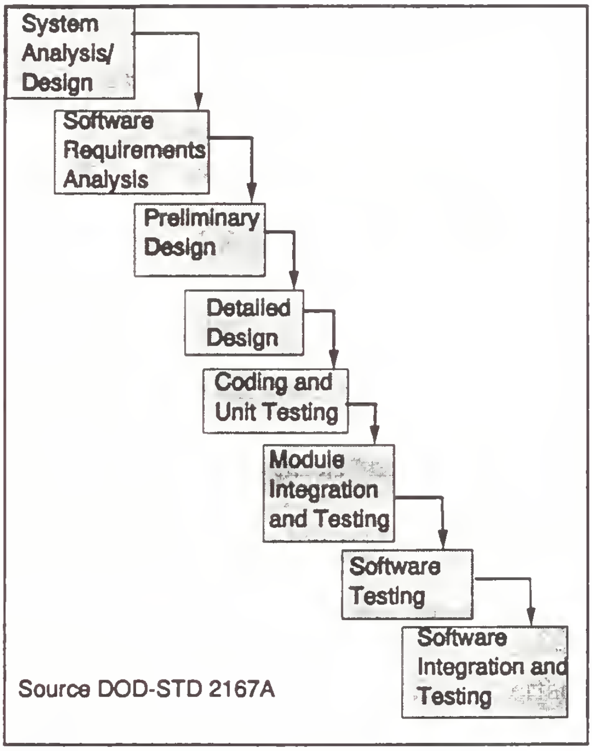


Figure 6 Waterfall model of software development

The cycle starts during initial system design when the hardware and software are not yet separated. Once system design has been completed in the development cycle, the hardware and software must be integrated. This is because the two are usually developed along separate lines. Before the system goes into its test and evaluation period, the hardware and software are integrated into the system configuration.

The first step in the development process is to generate the system-level requirements. These requirements are reflected in the System/Segment Specification

(SSS), commonly referred to as the Type A specifications. The specifications at this time are neither hardware- nor software-specific.

After the Systems Requirements Review has been completed, actual system design begins. The purpose of the system design process is to establish the functional baseline. The initial subsystem/segment designs are also developed at this time. Finally, the systems engineering practices that are going to be employed during the system development are refined. It is here we first see software-specific deliverables. (MCCR, 1990, ch. 5, p. 1-14)

a. Software Requirements Analysis

Once the functional baseline has been established, the design process proceeds into the Software Requirements Analysis phase. (DOD-STD-2167A, 1988, p. 5.2) Typical products of this phase include:

- An estimation of the size of the software effort.
- An estimation of the software support tools required.
- Prototypes of some high-risk areas.
- Final versions of the software specifications.
- Updated software development plan.

At the conclusion of the software requirements analysis phase, the last two items on the list are reviewed by Government representatives at the Software Specification Review (SSR). This review will produce the software allocated baseline for the project. (DOD-STD-2167A, 1988, p. 4.1-4.6)

b. Software Preliminary Design

After the software allocated baseline is established, the software developer will begin software preliminary design. During Preliminary Design the total software structure is determined, as well as the relationship between individual components of the software. Products that usually result from work during this phase include:

- The Software Design Document (SDD).
- The Interface Design Document (IDD).
- The Software Test Plan (STP) (DOD-STD 2167A, 1988, p. 5.1)

c. Detailed Design

The purpose of detailed design is to logically define and complete the software design. When the detailed design phase is concluded, all of the allocated requirements will be satisfied. The software must be designed to a level such that someone other than the original designer can finish the project. The functions of all components, inputs, outputs, and constraints will be defined. In addition, the relationships between different components will be specified. These relationships include logical, static, and dynamic relationships of all components. The component and system integration test procedures will also be produced. Typically, the products of this phase include: (MCCR, 1990, ch. 5, p. 1-14)

- A detailed description of the computer processing associated with the system.
- A detailed description of the data.

At the conclusion of detailed design, a Critical Design Review (CDR) is conducted by the Government. This review assures the customer that the software design meets the requirements of both the system level specifications and the software development specifications.

d. *Coding and Computer Software Unit (CSU) Testing*

Coding is the point where the detailed design is translated into a software program. The source program lists are generated during the coding phase When the coding is complete, the programmers check the source code for errors. When the programmer is satisfied that the source code correctly carries out the detailed design, he than compiles the program. Compiling a program translates the source code into a machine-executable form. (DOD-STD-2167A, 1988, p. 5.5)

The purpose behind Computer Software Unit (CSU) testing is to eliminate any errors that may exist because of the coding process. Errors occur for many reasons. They include errors due to programmer mistakes, deficiencies in the software requirements, or deficiencies in the design documentation. The responsibility for CSU testing usually falls on the programmer who coded the unit. For the software process to be efficient, a rigorous unit-level testing program must be implemented. (MCCR, 1990, ch. 5, p. 1-14)

e. Computer Software Component (CSC) Integration and Testing

The purpose of CSC Integration and Testing is to integrate the software units and components that have been tested independently, and test them as CSC's. The idea is to demonstrate that the combination of components will fulfill the system design. The conclusion of this phase is the integration of the individual CSC's. These items are tested at the Test Readiness Review (TRR) prior to starting CSCI testing. (DOD-STD-2167A, 1988, p. 5.6)

f. Computer Software Configuration Item (CSCI) Testing

Upon the successful completion of the Test Readiness Review, the developer performs CSCI testing. This is the last step in the software development cycle. It precedes the integration of the hardware and software portions of the system. The purpose of CSCI testing is to perform formal tests according to the software test plans and procedures that were previously developed. Testing at this point demonstrates that the software satisfies the Software Requirements Specification and the Interface Requirements Specifications. (DOD-STD-2167A, 1988, p. 5.7) When the software is released for integration testing, a Functional Configuration Audit (FCA) and the Physical Configuration Audit (PCA) are conducted. The software FCA is the Government's method of verifying that the CSCIs perform according to their requirements and interface specifications. The software PCA is a formal technical examination of the as-built software product against its design. (DOD-STD-2167A, 1988, p. 5.7)

g. *Systems Integration & Testing*

Systems Integration and Testing proves that the developed software will work once it is integrated into the system environment. The entire integrated hardware and software system is turned over to the Government after the Formal Qualification Review (FQR). The FQR is a system-level review that verifies that the actual system performance complies with the system requirements. It is at this point that the contractor's role will diminish. Configuration control of the software will revert to the Government once the product baseline has been approved. (DOD-STD-2167A, 1988, p. 5.8)

2. Post Deployment Software Support

The PDSS activities perform configuration control over software they support. There are three types of changes that the PDSS activities make: those caused by latent defects, those caused by a user-requested enhancement, and those required by a major product improvement. The PDSS activities usually perform the first two changes themselves, while they generally contract out the software changes necessitated by a major system upgrade. (MCCR, 1990, ch. 7, p. 1-11)

Software does not fail in the same way that hardware does. When hardware degrades overtime, its components simply wear out. A software problem occurs because of an error that existed since the software's creation or was introduced by subsequent maintenance. When a problem caused by a hardware component occurs, the problem is fixed by bring the component back to its original configuration. If the problem is a

software error, a new configuration is created when a problem is fixed. (MCCR, 1990, ch. 7, p. 1-11)

When a user sends in a software complaint, the Army maintenance directorate sends a logistic support representative assigned to the area to the user activity to investigate the complaint. Once the complaint has been verified and determined to be a software problem, it is sent to the PDSS activity. The PDSS activity tries to duplicate and identify the cause of the problem. Once the source of the problem is identified, solutions are developed and tested to ensure that the original problem has been solved. Regression testing is also done to ensure that no new problems are introduced during the software maintenance process. When initial system tests are completed, integration tests are then run on actual equipment to ensure that the software runs without any errors. Finally, an interoperability test is run. This test ensures that the software change has not created problems with any other piece of equipment with which the software must operate. The final step in the software maintenance process is to update all of the effected documentation. (MCCR, 1990, ch. 7, p. 1-11)

When the software change is completed and validated, it is distributed to the users. For low-density systems with limited distribution, the change may be hand delivered and installed to ensure smooth execution. For large-quantity, widely distributed systems that may require larger numbers of changes, the users are supplied through the distribution process with written instruction packages on installing the new software release. (MCCR, 1990, ch. 7, p. 1-11)

V. METHODOLOGY

This chapter discusses the methodology and types of data that were gathered during the research. Interviews with PDSS personnel were conducted both over the phone and in person during visits to PDSS activities at CECOM and MICOM. Topics include personnel, training, budget, operations, and senior leadership.

A. PERSONNEL

As part of DoD's effort to downsize the military, the civilian employees of DoD are undergoing much the same downsizing effort as the active uniformed force. The uniformed force has been given a vision of what it will look like in the near future, which is the ten-division Army. To date, no reference was found to a similar analysis performed on the civilian side of the Army. Rather than analyzing the processes that are being performed, it appears the Army has resorted to across-the-board cuts, hiring and promotion freezes, and incentives to leave Government service. In many cases only people have been reduced, without a commensurate reduction in the work load. The PDSS activities are largely manned by civilians. The following section addresses personnel issues.

- Number of personnel presently working in the PDSS centers.
- Number of Government civilians.
- Number of uniformed military.

- Number of contractor personnel.
- Rank structure of the PDSS centers.
- History on the number of Government personnel authorized back to 1985.
- History on the number of contractor personnel hired back to 1985.
- The extent of future personnel reductions.
- Impact the draw-down has already had on the PDSS centers.
- The use of contractors to make up for lost Government personnel.
- Effects that reduced resources have on the PDSS centers.
- Effects of further budget cuts on systems currently supported.
- Personnel turn-over rates at the PDSS centers.
- Personnel policies instituted at the PDSS centers.

B. TRAINING

The state of the art in software engineering is constantly changing. It is important that the PDSS centers have a training program in place to keep their employees at the leading edge of software engineering practices. Have budget cut-backs had an adverse effect on the training in the PDSS centers? Some of the issues that were addressed by this thesis follow:

- The education levels of Government civilians.
- The types of training the PDSS centers give.
- The impact of budget cuts on the PDSS centers ability to improve their processes.

C. BUDGET

The Federal Government, and DoD in particular, are in the process of down-sizing. Budget reductions are one effect. It was desired to gather a budget history of the PDSS centers, and compare the historical budgets with their projected out-year budgets. The following issues were addressed by this thesis:

- The budget history of the PDSS centers since 1985.
- Projected future budgets of the PDSS centers.
- The impact that the budget and personnel cuts could have on future software maintenance.
- The impact of unstable funding profiles.
- Implications of budget cuts.

D. OPERATIONS

In order to come to any reasonable conclusions with regard to budget data, it would be necessary to have a history of the software support for systems provided by the PDSS centers. The following issues were addressed by this thesis:

- History of the software supported by the PDSS centers since 1985.
- SLOC supported.
- Number of languages supported.
- The role of the contractor in the PDSS centers.
- How the PDSS centers classify and prioritize their workload.

- Status of the support equipment in the PDSS centers.

- The types of support the PDSS centers provided the troops in the field during Operation Desert Shield/Desert Storm.

- The impact the current round of budget and personnel cuts would have on the PDSS centers ability to provide the same level of support during similar potential conflicts.

E. LEADERSHIP INTERVIEWS

It was desired to obtain the opinion of the Director or Deputy Director of each of the PDSS centers on the following issues:

- The impact of existing budget and personnel reductions.

- The impact of the reduction in software maintenance support.

- How the PDSS center leadership handled budget and personnel cuts.

- The current concerns of the senior PDSS leadership.

VI. DATA ANALYSIS

This Chapter presents data gathered during the research process. In all, personnel from two centers were surveyed and visited. The data from each PDSS center is presented separately. In some cases, the data is consolidated in order to retain anonymity of the source.

A. INTRODUCTION

The following sections will present the history of the PDSS centers. In order to put all of the data in its proper context, the roles and differences between the two PDSS centers are examined.

1. History

All of the surveyed centers evolved in a similar manner. This evolution is characterized by a briefing obtained from the Communications Electronics Command (CECOM) Software Engineering Directorate (SED) (CECOM SED, 21 February 1994). It shows that the PDSS centers evolved through four stages. Although both PDSS centers evolved similarly, each had a somewhat different focus.

a. Stage One

Stage one of the PDSS evolution started prior to 1981. During this period, the only players involved in the software process were the program managers (PMs). Their primary emphasis was on hardware and software development, not total life cycle support which includes the maintenance phase.

Several problems resulted. They included an inadequate emphasis on life cycle support needs and little standardization in the processes created because of the lack of coordinated decision making. In addition, software expertise was spread across multiple PMs. This duplication of responsibility usually meant that software personnel were not able to obtain the training necessary to keep them skilled in current technology. It also meant that if a software professional wanted to get promoted, he had to move out of software management into another acquisition field. (CECOM SED, 21 February 1994)

b. Stage Two

Stage two of the process occurred between 1981 and 1983. The key players for this period were the PMs and the newly formed PDSS centers. The PM's primary emphasis was once again on software development issues. The PDSS centers were concerned only with providing PDSS.

This stage of the evolution process filled the support void for fielded systems that had been present during stage one. However, problems soon emerged in the transition of software from the PM to the PDSS activities. The major problem centered on the inability of the PDSS activities to significantly influence the software development process to consider life cycle support issues. (CECOM SED, 21 February 1994)

c. Stage Three

The third stage in the evolution of PDSS centers occurred from 1983 to 1985. The players involved during this period were the PMs and the PDSS centers,

which were renamed Life Cycle Software Engineering Centers (LCSECs). The LCSECs emphasis changed from conducting PDSS and expanded to include full life cycle support for software. This included involvement during development. PMs were still concerned primarily with software development.

This change in the LCSECs focus eased some of the problems neglected during stage two. It also improved the previously neglected process of planning for PDSS. (CECOM SED, 21 February 1994)

d. Stage Four

Stage four of the PDSS evolution started in 1985 and continues to the present. At this stage, the primary players continued to be the PMs and LCSECs. PMs were still concerned with the management of the software development for their systems. The LCSEC's broadened their mission to include addressing technical programmatic and developmental software issues during the development process. In addition, they also sought to incorporate life cycle support issues into the developmental process. PDSS activities were still concerned with PDSS planning, as well as performing PDSS.

This fourth stage of the evolution eliminated many of the software transition problems and provided a healthy base for standardizing interfaces across systems. Centralizing the software personnel at LCSECs also improved the career promotional opportunities for software personnel. (CECOM SED, 21 February 1994)

2. LCSECs Role in the Life Cycle

Under the current concept the LCSEC's are involved in all phases of a system's life cycle. During the initial developmental phases of a system, the LCSECs primarily provide functional and software program support. Functional software support involves providing software development expertise, such as verification and validation, or pre-award contractor surveys. Software program support involves the software personnel who manage and track the contractor's activities on a day-to-day basis.

During this period, the LCSECs software personnel provide technical support to both the program executive officers (PEOs) and PMs. They ensure that the management and technical decisions made during development address the PDSS needs of the software. The LCSECs also use this early developmental time to oversee and gain knowledge about the software design. The software design will play a key role in the maintainability of the software.

As software development progresses to coding and software integration testing, the LCSEC personnel provide counsel to the PM and play key roles in reviews and inspections defined in the contract.

During the production and support phases, the LCSECs are dedicated to the PDSS of the software. As such they focus on modifying, refining, and controlling the software. (CECOM SED, 21 February 1994)

3. Differences Between the PDSS Centers

There are some differences between the two LCSECs discussed in this thesis. These differences make it difficult to directly compare and contrast data between centers.

CECOM SED most closely fits the model discussed previously in Paragraph 2. They provide support for the entire software life cycle. At CECOM, the PMs obtain all their software management and functional expertise from the CECOM SED.

At MICOM SED, the previously discussed model does not entirely hold true. Like the CECOM SED, the MICOM SED is organized under the Research Development Engineering Center (RDEC). However, if a PM needs software management expertise (e.g., oversight of a particular software subsystem) he gets such support from a pool of software professionals who work for a separate division of the RDEC, not for the MICOM SED. A PM who needs pre-award contract surveys or verification and validation obtains such functional expertise from the MICOM SED.

B. PERSONNEL

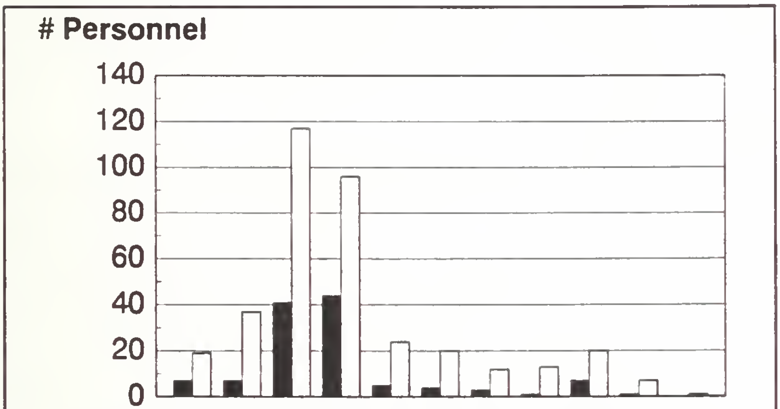
Personnel staffing at the SEDs consists of a combination of Government civilian employees (with a GS or GM designation), military personnel, and contractor employees.

1. Demographics of the PDSS Work Force

CECOM SED has a current on-hand strength of 366 Government civilian employees, 24 military personnel, and 1290 contractor personnel for a total work force of 1680. It provides support to 227 systems for a ratio of 7.4 individuals per system. (Wagner, 1993)

The MICOM SED has a current authorized strength of 128 personnel, with an on-hand strength of 103 Government civilians and 327 contractor personnel. The

MICOM SED also has 12 software interns presently working part-time who will start working full time April 1994. These software interns are not counted in their current strength figures. Although it has the authorization for two uniformed military personnel in its table of organization, the MICOM SED currently has none on hand. The military slots are desired and are projected to be filled in the future. The MICOM SED work force provides support for 47 systems. The total work force of 436 individuals provides a ratio of 9.3 individuals per system. (Rostollan, 1993)



Grade	15	14	13	12	11	9	7	6	5	4	2
MICOM ■	7	7	41	44	5	4	3	1	7	1	0
CECOM □	19	37	117	96	24	20	12	13	20	7	1

Figure 7 CECOM and MICOM SED's Grade Structure (Rostollan, 1993) (Wagner, 1993)

Both of the PDSS facilities have similar rank structures as can be seen in Figure 7. This rank structure is heavily weighted at the GS 12 and 13 level. There is limited potential for promotion above GS/GM 13.

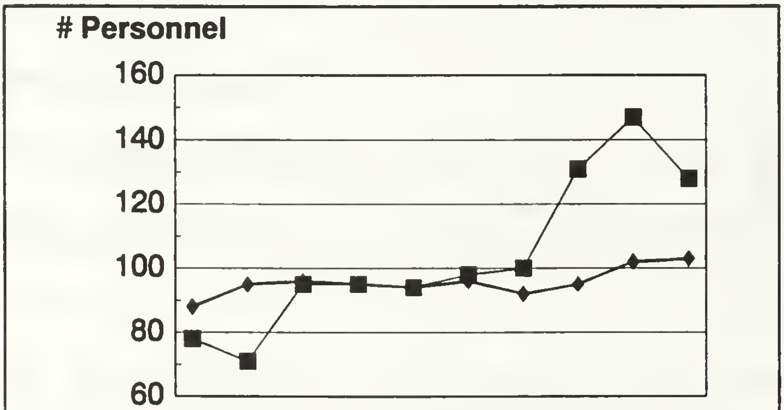
In the CECOM SED organizational structure, the individuals shown in Figure 7 who are higher then GS 13 are in charge of individual sections and serve in management assignments. Most who are below the GS 13 level hold technical positions.

As previously mentioned, the MICOM SED's rank structure is nearly identical to that of the CECOM SED. This rank structure places 71% of the engineers at a level

of GS 12 or 13. The positions below that are generally non-technical positions. Again, as was the case at the CECOM SED, there are relatively few high-level positions available for software engineers to advance beyond GS 13.

2. History of the PDSS Work Force

All of the LCSECs have undergone continuous change since their inception. This is not surprising given the constant increase in software to be supported, as well as the increase in the use of software in weapon systems. As a result, tracking the history of the number of individuals authorized in past years was difficult. Only one of the LCSECs had historical data on personnel.



MICOM	FY 84	85	86	87	88	89	90	91	92	93
AUTH ■	78	71	95	95	94	98	100	131	147	128
ON-BOARD ◆	88	95	96	95	94	96	92	95	102	103

Figure 8 MICOM SED Personnel History (Rostollan, 1993)

The MICOM SED kept very good records in this area. Figure 8 shows the MICOM SED's history of personnel growth from fiscal year 1984 through 1993. The MICOM SED has experienced significant growth since its inception. In FY 1985, it was authorized 71 personnel; this figure grew to a high of 147 in 1992. The growth in the number of authorized personnel coincides in part to the growth in the number of systems supported. Also, during the period the MICOM SED began an over-hire policy. This

allowed the MICOM SED to hire additional temporary Government employees to fill increased system support needs. In 1993, the over-hire policy was abolished, and the number of authorized personnel dropped to 128. The abolishment of the over-hire policy occurred because of budget cuts in the MICOM RDEC.

An interesting trend can be seen by looking at the number of personnel that the MICOM SED has assigned. In 1984, the SED started over-strength in actual versus authorized personnel. This soon changed, and for a period of several years their on-hand strength closely tracked the authorized strength. In 1991, because of the increase in the number of systems supported, the number of authorized personnel rose from 100 to 131. The number of people on hand stayed nearly constant with the year before figures. From 1992 through 1995 there was only a slight increase. This gap can be primarily attributed to the hiring freeze in place at the MICOM SED. These numbers do not reflect the software interns that the MICOM SED had on hand during the periods covered.

3. Future Projections for the PDSS Work Force

The CECOM SED's personnel authorization has been cut from its current level of 366 Government civilians down to 250 Government civilians as part of CECOM's effort to downsize its personnel force. This is primarily attributable to the overall reduction in the size of the Army work force. This reduction forewarns of significant PDSS software personnel support shortfalls for the systems assigned to the CECOM SED. The CECOM SED currently projects it will meet its draw-down figure by early retirements and through the DoD personnel buy-out incentive program. They do not

project having to undergo a reduction in force (RIF). However, further personnel cuts could require forced reductions. (Wagner, 1993)

Reductions in Government personnel are often offset by an increase in the number of contractor personnel hired to handle increased work loads. However, the CECOM SED has recently lost funding support for four of the systems currently under PDSS support. Therefore it is unlikely the reduction in Government personnel will be offset by contractor personnel.

Discussions with MICOM SED personnel indicate that they have been spared further cuts in the current round of manpower cuts. The MICOM RDEC appears to understand the need for the MICOM SED to be shielded from Army downsizing cuts. Personnel at the MICOM SED indicated there may even be some personnel growth in the near future. (Craig, 1994)

4. Examples of Problems Because of Personnel Cutbacks

From discussions with both LCSECs, the loss of personnel has had no direct affect on the PDSS mission in the short run. To a large degree this is because of the large amount of work that is performed by contractor personnel. Only budget cuts have an immediate and detrimental effect on the center's ability to perform their missions.

An example of the problems experienced because of personnel cut-backs has to do with the loss or reassignment of experienced personnel when a system's funding is terminated. The TSQ-73 is a battalion- and brigade-level missile manager for the HAWK air defense system. The TSQ-73's PDSS funding was cut as part of the FY 1994 budget deliberations. The MICOM SED has several contractor personnel who have worked

exclusively on that system for over 20 years. Some have worked for as many as seven different support contractors. With the TSQ-73 funding cuts, such individuals will either be reassigned or terminated. For software maintenance, knowledge of the design and structure of the software is crucial and a major productivity factor. Such knowledge is not quickly accrued. Should the Army at some time in the future decide to reestablish PDSS support for this system it is likely that it will not be able to rehire these experienced and knowledgeable individuals. Any attempt to reestablish PDSS support for the TSQ-73 will take significant time to reach past productivity and quality levels. (Rostollan, 1993)

Many do not understand the cognitive challenge software maintenance involves. Most would accept that an experienced automobile mechanic can easily adapt and perform most maintenance tasks on just about any automobile. Many presume the same is true for those performing software maintenance. After all, it is maintenance. This simply is not the case. Those involved with software maintenance have one of the most intellectually challenging jobs in our society. Studies have shown that upwards of 30% to 50% of a software maintainer's time is spent just trying to understand the design and unsort what others have previously done to the code. It takes new programmers anywhere from six months to one year before they are fully productive maintainers. Army leadership should review the significant consequences personnel cuts have on software support. When PDSS for a system is terminated, serious consideration should be given to removing the system from the field.

Another problem confronting the PDSS activities is the mix of contractor to Government staffing levels. The CECOM SED indicated that the optimal range for contractor staffing of PDSS activities is between 50% and 60% of the overall effort. Currently both LCSECs have contractors providing in excess of 70% of the support effort. Retirements, transfers, and buyouts of Government personnel are likely to drive up the contractor-to-Government ratio as more contractors are hired, contingent on the availability of funding.

This raises the serious issue of the role of Government personnel in PDSS. As the percentage of Government employees continues to decrease, they more and more assume the roles of oversight and administration. Fewer are directly involved in software engineering functions and the Governments's knowledge and understanding of the software deteriorates. This is another serious problem Army leadership should address. (CECOM SED, 21 February 1994)

5. Personnel Policies

Both of the PDSS facilities are under strict hiring freezes as part of the Government's effort to downsize. Both the CECOM and MICOM SEDs currently experience a Government personnel loss rate that runs between 10% and 12% per year. This equates to an annual loss rate of 24 individuals per year at the CECOM SED, and 11 individuals per year at the MICOM SED. (Doughtery, 1993) (Rostollan, 1993)

The LCSECs use two methods to offset this loss of personnel. The first method is with the use of software intern's. The second method is through the

consolidation of a mission area, such as the consolidation of the avionics support mission from the ATCOM to the CECOM SED.

Under the software intern program, the Government hires individuals (generally electrical engineers) straight out of college. These individuals attend a one-year software engineering internship, followed by half-day attendance at a fully-funded masters degree program in software engineering. A more detailed discussion of the software intern program is presented under the training section of this report. Both the CECOM and MICOM SEDs said they each received 10 to 13 people a year from this program. (Doughtery, 1993) (Rostollan, 1993)

A second method that the LCSECs have to get more people is through consolidation of PDSS missions. The CECOM SED indicated they were in "a state of perpetual reorganization." As they pick up new missions, or as software functions are centralized, they pick up additional people and work. An example of this is when the CECOM SED received the avionics support mission from the ATCOM PDSS activity. (Wagner, 1993)

The Government's attempt to cut the federal work force by buying out individuals has also had an effect on both activities. Although the buy-out currently does not apply to software engineers, it does apply to other key specialties such as engineers, computer scientists, and mathematicians. The buy-out along with early retirements are both methods the CECOM SED is using to achieve its downsizing goal. (Turner, 1994)

Presently, none of the SEDs indicated they are losing personnel to commercial industry. However, this may in part be only due to the poor economy the past couple of

years. In addition, a large number of software engineers have entered the job market because of the downsizing of the defense and aerospace industries. If the national or local economies start to pick up, there is concern the LCSECs could start experiencing significant personnel loss to industry due to the limited promotion opportunities and the pay differential with the private sector.

The final way the SEDs lose people is through promotions and transfers. The SEDs, besides being under a hiring freeze, are also under a high-level promotion freeze. If an individual wants to get promoted, he must normally leave the SED and move to a job outside the SED. In the case of the MICOM SED, individuals were moving to the PEO office where they could get promoted. Now, the MICOM PEO structure is undergoing a RIF. This has largely dissipated this avenue of loss. Most Government employees are staying put and trying to ride out the RIFs and downsizing. (Doughtery, 1993) (Rostollan, 1993)

C. OPERATIONS

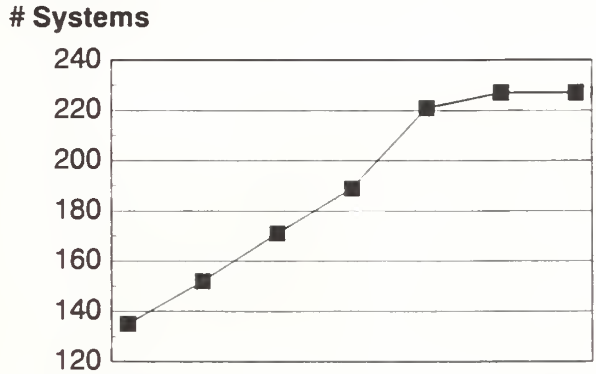
The following paragraphs discuss PDSS center operations. Among the topics addressed are the support software systems and the role contractors play in PDSS.

1. History of Software Supported

Figure 9 shows the growth in the number of systems that the CECOM SED has experienced since 1988. Its records show a continual increase in the number of systems for which they are providing support. The number of supported systems for FY 1994 has stayed constant only because four systems supported in past years lost OMA

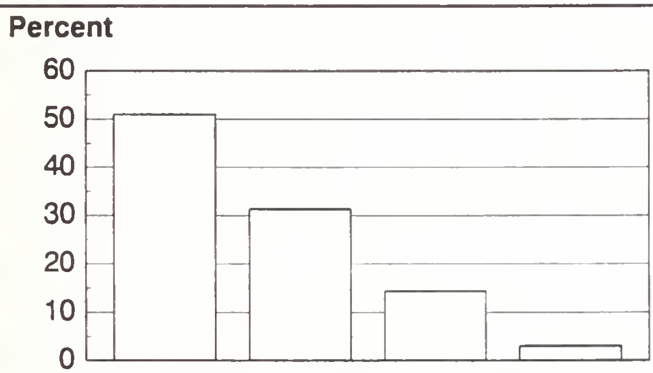
funding and support was cancelled.

The CECOM SED could not show historical records for the total number of source lines of code (SLOC) that it supports. However, Figure 10 divides up the 227 supported systems, and arranges them by the number of SLOC. Because the complexity increases dramatically with larger programs, the software maintenance work load increases significantly as the SLOC in a system rise. Figure 11 shows a similar system growth pattern for the MICOM SED. It has also experienced a continual rise in the number of systems it



CECOM	FY 88	89	90	91	92	93	94
SYSTEMS	135	152	171	189	221	227	227

Figure 9 CECOM SED History of Systems Supported (CECOM SED, October 1993)

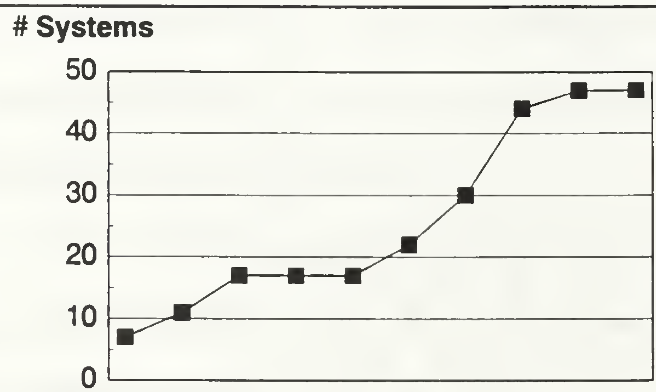


	0-100	100-500	500-1000	1000+
Percent	51.0	31.4	14.4	3.1

In thousands of SLOC

Figure 10 CECOM SED Current System Breakout by Number of SLOC (Wagner, 1994)

supports. This growth has only recently begun to level off. This is primarily because budget cuts have slowed the introduction of new systems.

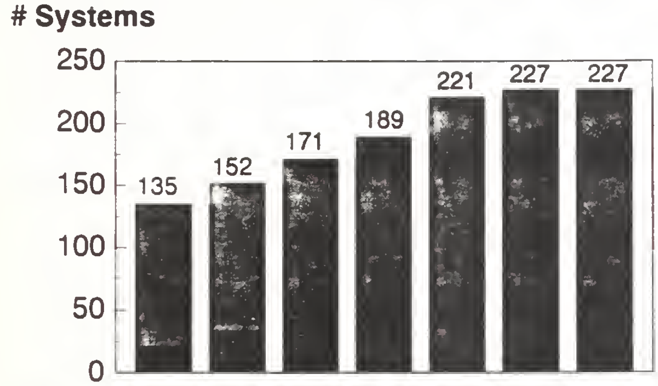


MICOM	FY 84	85	86	87	88	89	90	91	92	93
SYSTEMS	7	11	17	17	17	22	30	44	47	47

Figure 11 MICOM SED History of Software Systems Supported (Rostollan, 1993)

The MICOM SED was not able to provide a count of the total number of SLOC for which they provide support. While using lines of code versus the use of function points as a measure for the last several years has been debated, the fact that there is no available measure other than number of systems presents a problem when trying to assess the work load for an activity.

Figure 12 depicts a future problem that potentially is even more critical for the LCSECs. That is the difference between the number of systems that are currently fielded and in PDSS, and the number of systems that are in the development phases of the life cycle. While the data presented in Figure 12 is from the CECOM SED, similar situations hold for the other Army centers. The vast majority of systems that the LCSECs support are not in the PDSS phase of the life cycle, but rather some phase of new system development. There is a significant number of systems that are getting ready to deploy and move into the PDSS phase of their life cycle. Even if only 50% of these systems are



	FY 88	FY 89	FY 90	FY 91	FY 92	FY 93	FY 94
OMAPDSS	21	34	43	48	54	62	65
MATRIX	114	118	128	141	167	165	162

Figure 12 CECOM SED History OMA and Matrix Supported Software (CECOM SED, October 1993)

eventually fielded and require PDSS support, the number of systems requiring PDSS would more than double. These systems will be manpower intensive and require a significant increase in the number of personnel (either Government or contractor) necessary to adequately support them.

2. Role of the Contractor in the LCSECs

The contractor plays a key role in both LCSECs that were visited. In both cases, the centers paid out over 70% of their budget to fund contractor-performed work. The centers perform all of their software support work in a modularized form. This enables them to contract for the support for individual systems to contractors. These contracts are worth millions of dollars annually. Some of the contractors that are involved in providing this support for Army LCSECs include:

- TRW.
- Scientific Applications International Cooperation.

- Rockwell International.
- Hughes Aircraft Company (Craig, 1994).

These are all well-know defense contractors. Most of the work performed by the two LCSECs is performed on site. One advantage of using on-site contractor personnel cited in discussions was the ability of the Government to gain access to high quality software personnel at a lower cost. Should the contractors perform this same work at their company locations, the Government would be forced to pay high overhead costs on facilities.

3. Classification and Prioritization of the Work Load

The LCSECs both indicated that they get very little discretionary funding. If a PM wants work done on a system, he provides the LCSECs funding and they will work on it.

For software in the PDSS phase of the life cycle, the LCSECs, in conjunction with the PM, prioritize the changes for each system, along with the projected costs for those changes. The changes are generally divided into three categories: "must change," "need to change," and "would like to change." Once the list is compiled, it is sent to AMC for review. From there, Training and Doctrine Command (TRADOC) rank orders/prioritizes the systems. As a set amount of OMA funding has already been determined, everything that falls above the available funding line gets funded, while everything below it gets canceled. As unsophisticated as the system is, at least the Department of the Army is attempting to address the funding shortfall problem. Prior to

1993, there was no prioritization system, and the LCSECSs themselves had to make the work decisions based upon the funding they received. (Wagner, 1994)

The bottom line is the Army faces a software support crisis. There is not adequate funding to support all the fielded system's software.

4. Support Equipment

Both of the LCSECs visited had extensive computer and hardware resources. This equipment includes the necessary computers and peripherals for conducting software support. In many cases the actual tactical equipment/system used in the field is used for testing software and firmware changes and trouble-shooting software problems. The centers have the capability to integrate software changes onto actual equipment and onto actual nets on which the equipment will be used. In the CECOM SED's case, this has allowed it to perform software maintenance and enhancements on not only Army software, but also on Marine and Air Force equipment that interface with Army tactical communication systems.

5. Operation Desert Shield/Desert Storm Support

Everybody is familiar with the sight of the Patriot missile system engaging incoming SCUD missiles in Saudi Arabia during Operation Desert Storm. During the operation key software changes were made to the Patriot system. Both the CECOM and MICOM SEDs were heavily involved in providing software support to many systems of units deployed in Operation Desert Shield/Desert Storm.

The MICOM SED, along with Raytheon Corporation, provided software support for the Patriot units deployed during Desert Shield and Desert Storm. Because the Patriot missile system is still undergoing major modifications, the system's PDSS support is still provided by the Raytheon Corporation. The MICOM SED provides functional oversight for the PM. Examples of the kind of work that the MICOM SED performed:

- Ran tests on special software deviations/waivers for the Patriot missile system during Desert Shield and Desert Storm.
- Ran formal qualification regression tests of Desert Shield and Desert Storm software builds.
- Supported testing of deviation/waivers at the White Sands Missile Range.
- Ran tests using the PATRIOT radar to investigate radar problems from Desert Shield and Desert Storm.
- Performed independent analyses of deviation/waiver tests run by Raytheon.
- Participated in the Joint Analysis Team review of Desert Shield and Desert Storm data in Huntsville.
- Performed analysis of missile guidance data at Raytheon, Boston.
- Supported radar testing at White Sands Missile Range.
- Supported special missile flight tests at White Sands Missile Range.
- Produced charts and reference material across all functional areas to aid the PATRIOT Project Office trip to Israel.
- Analyzed problems and areas of concern throughout the Desert Shield and Desert Storm conflict that included such topics as cease fire, threat classification, target fragmentation, false detections, and fire unit correlation.
- Performed a parametric study of the safe passage corridor logic for the system.

- Studied tracking capabilities of the Patriot radar versus artillery shells.
- Performed a special review of all anti-tactical ballistic missile deviation/waivers.
- Studied debris impact point prediction algorithms. (Rostollan, 1993)

Besides the verification and test activities for the Patriot missile system that are listed above, the MICOM SED was responsible for generating several scenarios with tactical ballistic missile's (TBM) as targets. These TBM scenarios were used in the testing of the tactical software, as well as for training. (Rostollan, 1993)

Although the Patriot missile system was one of the most publicized systems that was deployed during Desert Shield/Desert Storm, many other equally important software-intensive systems were deployed. The CECOM SED supported many of these.

They also provided support to not only Army systems, but also joint service systems. For example, the SB-3865, Unit Level Circuit Switch, was used by Marine Corp units. When the Marines arrived in Saudi Arabia, they operated as a separate joint task force. This required that they have the ability to communicate to all the other service units and commands through the tactical satellite system. However, the SB-3865 had not been designed to operate in that mode. The system soon overloaded when demand exceeded the system's capabilities. The CECOM SED was asked to provide a software modification to fix the problem. Able to replicate the software failures at their location, they designed a software solution that was quickly implemented and resolved the deficiency.

Another important project during Desert Shield/Desert Storm that the CECOM SED provided support for was the AN/APR-39(V)2. The AN/APR-39(V)2 is a radar warning receiver installed on all U.S. fixed and rotary wing aircraft and used to detect radar-based threats to airborne platforms. The receiver is threat-based, and is only effective against threats that are incorporated into its database.

The CECOM SED has developed a rapid reprogramming capability for this system that includes threat analysis, electronic warfare environment simulation, and testing and programming capabilities. This has reduced the time required to analyze new threats and incorporate the necessary software changes into the AN/APR-39(V)2's threat database. (CECOM SED, February 1994)

Because the Iraqis were operating a large number of aircraft also operated by the coalition forces, the system had to be upgraded. For example, both the French and the Iraqis used the Mirage F-1 aircraft. With France being a U.S. ally, the Mirage F-1 had never been entered as a threat. From August 1990 to November 1990, the CECOM SED performed four separate threat analyses and reprogrammed the AN/APR-39(V)2 based on new threat scenarios. (CECOM SED, February 1994)

The following is a typical example of the software support effort performed by the CECOM SED during Operation Desert Shield/Desert Storm. On 8 August 1990, the CECOM SED received 26 new threats against the AN/APR-39(V)2 that needed to be analyzed. Once the analysis was complete, electronic warfare simulation and testing was conducted on the AN/APR-39(V)2 test-bed and the software was modified. On 18 August, just ten days after receipt of the threats, the analysis, programming, and testing

were completed and the modified software distributed to units in the field. Such effort was responsible for the improved effectiveness of a large number of airborne platforms employed in Desert Shield/Desert Storm. (CECOM SED, February 1994)

Another example of the support provided during the war includes on-site personnel assigned to units in Saudi Arabia to track software problems. Having software knowledgeable personnel available to witness system problems first hand proved invaluable in accurately identifying and expediting the resolution of the many problems raised during the operations. During Operation Desert Shield/Desert Storm the CECOM SED had between one and two military personnel from their office stationed in Saudi Arabia at all times. They provided direct support for identifying software problems and forwarded the information to the CECOM SED. (CECOM SED, February 1994)

CECOM SED's timely support has not just applied to Operation Desert Shield/ Desert Storm. During Exercise Dynamic Guard 93 operations, Air National Guard (ANG) units were activated and deployed from their continental United States locations to remote sites in the Turkish mountains. These units deployed with AN/TSC-93A Tactical Satellites and SB-3865 Unit Level Circuit Switches. This equipment was critical to an elaborate communications network that included establishment of communication with an AN/TTC-39D/PS located at the NATO airbase in Corlu, Turkey. From this link, communications via tactical satellite to the 7th Signal Brigade Headquarters in Karlsruhe, Germany were to be established. This would allow access to military telephone lines.

Given the complexity of the interoperability required, it should not seem unusual that problems arose. Headquarters 7th Signal Brigade had learned that the

AN/TTC-39D/PS would not operate with the SB-3865. The CECOM SED received a request for emergency software support at 2:00 p.m. Friday 17 September 1993. (CECOM SED, February 1994)

At 4:00 p.m. the CECOM SED started building an emergency release tape for the AN/TTC-39D/PS. It contained a prototype software fix for the interoperability problem. While the emergency software fix was being prepared, arrangements were made to hand carry the emergency software release to Turkey in order to expedite that fix. (CECOM SED, February 1994)

Software engineers worked through the night and by 8:00 a.m. Saturday, 18 September, they completed building the emergency release tape. The fix was installed in the AN/TTC-39D/PS in the SED test bed and interoperability testing was conducted with the SB-3865 as well as other switches. By 10:30 a.m. the interoperability testing was successfully completed and a courier departed for the airport. (CECOM SED, February 1994)

The courier arrived in Istanbul, Turkey at 3:30 p.m. Sunday. A soldier from the 7th Signal Brigade stationed at the Corlu NATO Airbase met the courier and took receipt of the tape. By 5 p.m. the AN/TTC-39D/PS was up and operational. The problems were corrected. The ANG units arrived the following morning and successfully completed Exercise Dynamic Guard. While situations such as this are not every day occurrences, this example provides some insight on the demands made of SEDs. It demonstrates how critical software shortfalls can be to Army operations. (CECOM SED, February 1994)

D. BUDGET

As was mentioned earlier, funding is the only resource item whose reduction has an immediate effect on the operations of the LCSECs. Both of the LCSECs presently have greater than 70% of their operation's support provided by contractor personnel. A reduction in their funding translates almost directly to a reduction in contractor software support personnel.

Funding is the most unstable and erratic resource involved in software support. Besides the uncertainty in funding availability in general, the different types of funding and when they can be used must also be considered. The complexity for organizations trying to manage this process is incredible.

Software (like all other DoD items) is funded based on its location in the acquisition life cycle. Once a system's fielding is completed, its funding support comes from the Operations and Maintenance Army (OMA) account. A system under production is supported using procurement funds, while a system under development is funded using research and development funds. A fielded system may use research and development funding if such funding is used for upgrades. Finally, it is possible for different components of a system to be in different phases of the acquisition life cycle. While some elements are still under development, other components will have completed development and been fielded. As the individual components of a system complete their development runs, further software support is transferred to the LCSECs, if no major software upgrades are immediately planned. It is illegal to fund the different categories of support with an improper type of funding. A PM whose system is still using

procurement money cannot spend that type of funding to perform PDSS support on a piece of system equipment that has moved to OMA support. Under present funding reductions this is causing problems.

For example, the Army Tactical Missile System (ATACMS) has a missile test set associated with the system. The deployment of this test set has been completed. Its support has been transferred over to the MICOM SED. The ATACMS missile itself, however, is still under production and its software is funded using procurement funds. The PDSS support for the missile remains with the contractor because the missile is still in production and is scheduled to undergo several major system modifications. If the PM decides to enhance the test set, he will likely have to redevelop it under a system-wide product improvement instead of hiring the MICOM SED to perform the software enhancement. (Craig, 1994)

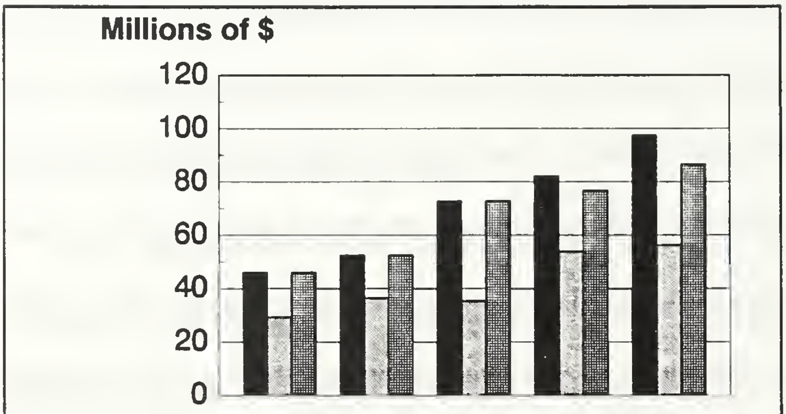
The reason that this creates a challenge relates to the Army's current method of allocating OMA funds for PDSS support. Prior to 1993, the LCSECs budgeted for OMA PDSS funds as part of their annual budget request. Based on the annual funding the LCSECs received, they notified the CINCS if a system's software support was terminated because of a lack of OMA funding. Typically, the CINCs would then lobby DCSOPS and AMC to continue system support and funds were usually found.

Starting FY 93 the LCSECs started submitting a budget request to AMC that stated how much it would cost to perform PDSS support on each of the systems they supported. The Army, which had a predetermined amount of OMA funding set aside for PDSS support, had TRADOC rank the list of systems in order of system priority. Once the

systems were rank ordered, funding was allocated by starting at the top of the list and proceeding until no more funds were available. All systems that fell below the cut-off line had their PDSS funding terminated. Under this new practice, DCSOPS then collects feedback from the CINCs on the impact of terminating PDSS support for a system. At the start of FY 94, 26 systems under PDSS support had their funding terminated.

An example of the reductions caused by such budget cuts is the AN/MSC-64, a satellite communication system used by the regional CINCs to maintain communication with the National Command Authority. Its funding was cut during the FY 94 budget process. However, based upon concerns raised from the CINCs, its funding was restored. There was enough negative feedback that DCSOPS was able to set aside more OMA funding to cover its PDSS support. This is a rather haphazard method of determining the funding status of a system, but reflects the results of budget cuts. However, not all systems have the visibility and political backing that the AN/MSC-64 has. Such systems have had their support terminated. (Wagner, 1994)

A major difficulty LCSECs encounter is the uncertainty of their budgets. Figure 13 illustrates this problem. This chart depicts three budget categories. Total required (TOT REQ) is the budget the CECOM SED requested to maintain the software for the fielded systems it supports. Initial programmed (INT PBG) refers to the funding it initially received as a result of the budget process. Total received (TOT REC) is the funding it actually received by the end of the fiscal year. While the data in Figure 13 was obtained from the CECOM SED, the basic challenge applies to all Army LCSECs.



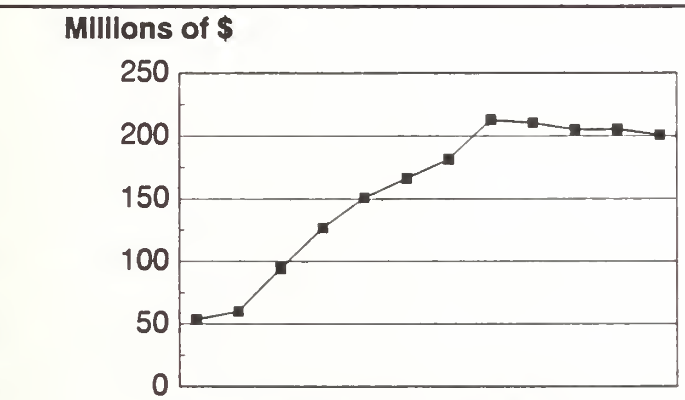
FY	89	90	91	92	93
TOT REQ	46.0	52.5	72.9	82.3	97.6
INT PBG	29.3	36.5	35.4	53.9	56.3
TOT REC	46.0	52.5	72.9	76.9	86.7

Figure 13 CECOM SED Budget History (Wagner, 1994)

The graph depicts a problem the LCSECs currently experience. They justify the work that they perform, and as the chart indicates, funding is requested at a certain level (TOT REQ). At the end of the budgeting process they are invariably funded at a much lower level (INT PBG). Their budgets are then incrementally increased over the fiscal year until they reach some unknown point (TOT REC). For example, in 1992 CECOM SED requested $82.3 million. By the initial program budget guidance it was allotted $53.9 million. At the end of the year the CECOM SED had eventually received $76.9 million, an increase of over 42%. Such unplanned growth presents a serious challenge to management.

In the past, as the year progresses, the LCSECs have usually ended up receiving almost all of the money they needed. However, starting about FY 91, that trend changed. Although the CECOM SED received funding above its INT PBG, it did not obtain all the funding it required to fully support all of its systems. Some support that was initially programmed was terminated, while enhancements on other systems were stretched out to a later date. This chart also illustrates the significant growth in software systems

requiring support. From FY 1989 to FY 1993 the CECOM support budget more than doubled.



FY	88	89	90	91	92	93	94	95	96	97	98	99
REQUIREDFUNDING	53.6	60.0	94.2	126.9	151.0	166.4	181.5	212.8	210.4	205.0	205.4	200.6

Figure 14 CECOM SED Projected Funding (Doughtery, 1993)

Figure 14 shows the history of the CECOM SED budget from FY 88 as well as projected required funding out to FY 99. Because of numerous reorganizations, the CECOM SED's budget figures could only be traced back to FY 88. These figures project an extraordinary growth in the budget, from $50 million in FY 88 to a projected requirement of over $200 million in FY 95. This is primarily due to new systems that will be fielded during the period and the resulting software support that will be required.

Figure 15, shows the history of the MICOM SED budget. Its budget showed a continual increase starting in 1984. This trend of increased software funding continued until reaching a high of $58.2 million in FY 88. Since 1988, the MICOM SED's funding levels have dropped significantly, from close to $60 million to $35 million in FY 93. These decreases in funding are commensurate with the decreased funding levels experienced by the program management offices located at MICOM.

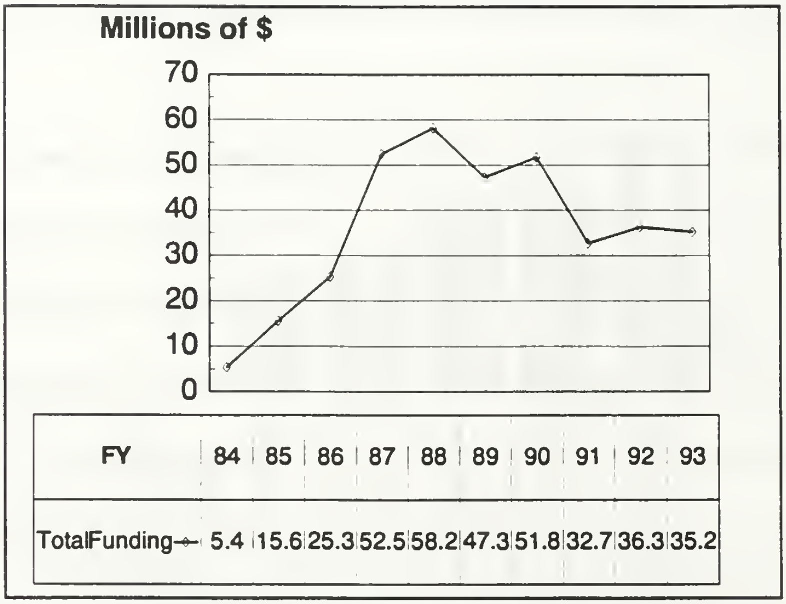


Figure 15 MICOM SED History of Funding (Rostollan, 1993)

The LCSECs are hardware intensive operations which have extensive mainframe computer operations. As these computers grow old, maintenance down times and costs become excessive (in many cases the manufacturers quit supporting them). The LCSECs must eventually replace these out-of-date or old pieces of equipment.

The MICOM SED obtains funding for new equipment from two sources. The first is from overhead charged to customers. The second source comes from directly charging the appropriate PM for equipment necessary to support their system. It first uses money accrued from the overhead charged to customers for their support. If this account cannot pay for new equipment, than the difference is charged out to the PMs based upon their usage of the equipment. (Craig, 1994)

The CECOM SED budgets for new equipment differently from the MICOM SED. It budgets for new equipment in the procurement (PA) account portion of its budget. Figure 16 shows the funding profile for this account. Prior to FY 90, the CECOM SED received steady funding in the PA account. During FY 90 the PA portion of its budget was reduced, and continued to decline to a low of $150,000 during FY 91. This occurred

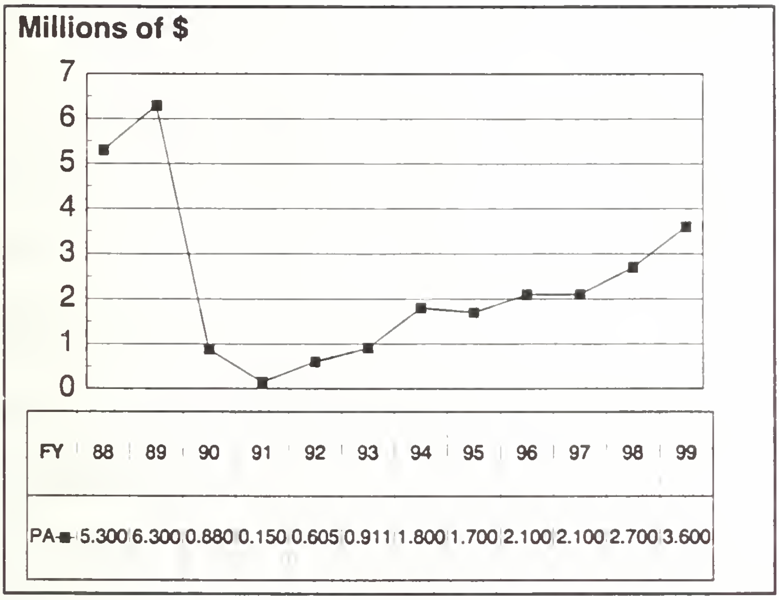
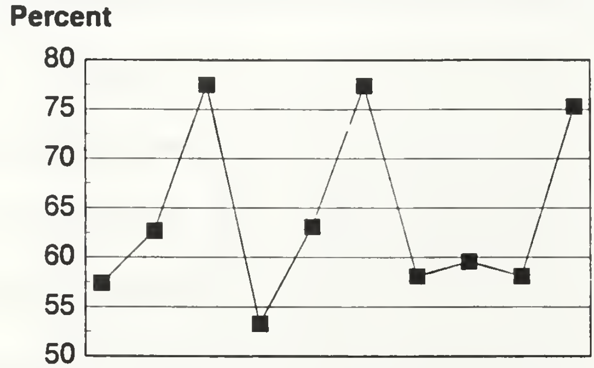


Figure 16 History of CECOM SED PA Account (Wagner, 1993)

because higher headquarters elected not to support the previous levels as part of an effort to reduce its own budget. More recently the procurement budget has risen somewhat. This is primarily because of a significant increase in demand for new equipment and a rise is expected to continue over the next several years. However, it should be noted that even during the next five-year budget estimate, the CECOM SED's procurement account is not expected to rise to the funding levels of the late 1980's. Although the CECOM SED has been able to live within the reduced funding levels, it has significantly reduced its modernization efforts. The risk to funding equipment modernization in this manner is that the equipment will grow old, and difficult as well as expensive to maintain. This problem could grow worse as a large backlog of equipment requiring modernization or replacement builds up. (Wagner, 1993)

Because of data limitations, it was not possible to provide a history of the percentage of the CECOM SED's budget that goes toward hiring contractors. Contractors currently make up 77% of the CECOM SED's work force. (Wagner, 1993)



MICOM	FY 84	85	86	87	88	89	90	91	92	93
PercentContracts	57.4	62.7	77.5	53.3	63.1	77.4	58.1	59.8	58.1	75.3

Figure 17 History MICOM SED Contractor Participation (Rostollan, 1993)

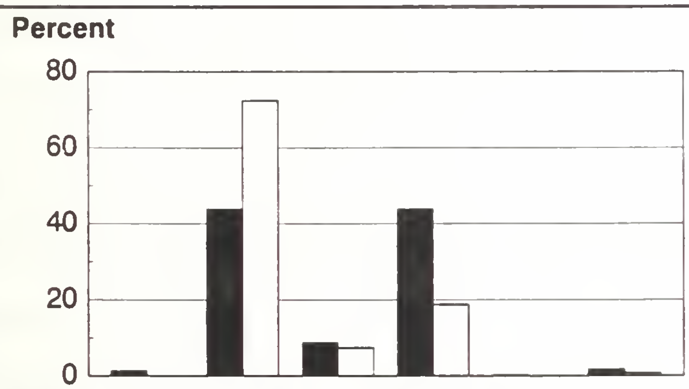
The history of the MICOM SED's contractor costs as a percentage of their budget overall is shown in Figure 17. The MICOM SED has a contractor ratio similar to that of the CECOM SED. Currently 75% of the MICOM SED's work is performed by contractor personnel.

Software development and support is a manpower-intensive effort. As such, the productivity and quality of the work effort to a large extent depends on the quality and continuity of the work force. The kind of erratic behavior depicted in Figure 17 creates serious problems. Under such conditions the contractor typically experiences significant turnover. New personnel normally take from six months to a year to become proficient. The quality of the personnel hired is also called into question. Top software engineers are unlikely to take a job given the uncertainty in job security. As a result such areas as software productivity, quality, and documentation all tend to degrade. Such problems also mean that software support schedules will slip. (CECOM SED, October 1993)

E. TRAINING

The state of the art in software engineering is constantly changing and advancing. In order for the LCSECs to remain competitive and provide the support that PEOs and PMs require, it is important that Government software professionals be as well trained as their counter-parts in industry.

1. Education level of PDSS personnel



Degree	Pur Bach	Bach	Pur Mast	Mast	Pur PhD	PhD
CECOM ■	1.4	43.9	8.8	43.9	0.3	1.8
MICOM □	0.0	72.4	7.5	18.8	0.0	0.8

Figure 18 Educational Levels of the LCSECs (CECOM SED, 21 February 1994) (Rostollan, 1993)

Figure 18 presents the educational levels of the personnel at the two LCSECs. The first category represents personnel who are pursuing their bachelors (Pur Bach) degree. Only a small percentage of the people fall into this category. The second category contains all individuals who have a bachelors degree (Bach). The third category is comprised of individuals who are pursuing a masters degree (Pur Mast). The fourth category encompasses all individuals who have a masters degree (Mast). The fifth category contains all individuals who are pursuing a doctorate. (Pur PhD) And the sixth category includes all individuals who have doctorates (PhD). All the individuals without a bachelors degree at the CECOM SED are enrolled in formal degree programs. At the

CECOM SED, 46% of the personnel had a masters degree or better. At the MICOM SED almost 20% of the personnel had a Masters degree or better. The educational levels at both LCSECs should continue to rise as long as the software intern program remains in place. (CECOM SED, 21 February 1994)

The software intern program, mentioned earlier in the chapter, is the primary method the LCSECs use to keep their education level high. Individuals selected for this program spend their first year attending an engineering intern program at Red River Army Depot. Their second year is spent at either Fort Monmouth or Redstone Arsenal. During this second year they work part time at the SED and attend school part time. They earn a masters degree in software engineering. Upon completion of the program, they owe the Government three years of service. Discussions with the CECOM SED indicated that the software intern program for FY 95 had not yet been funded. However, this is not unusual. For the last several years the decision to fund the software intern program has been made in the August time frame, just four weeks before the interns are supposed to report to Red River Army Depot. Personnel at the CECOM SED indicate that they expect the program to be funded this year. (CECOM SED, 21 February 1994)

2. Advanced Training in the Activities

The LCSECs use a variety of methods to train their personnel. Methods include the use of software-specific training, SEI improvement programs, and system specific user training.

The LCSECs provide training in a variety of areas that include:

- Total quality management.
- Software Engineering Institute standards.
- Radar operations.
- Software management.
- Software development.
- Ada program language training.
- Cost modeling.
- Object oriented software design.
- Computer architecture modeling.
- Budget management.
- Acquisition. (Rostollan, 1993)

As part of the LCSECs effort to improve their software development processes, each has undergone the Software Engineering Institute (SEI) software capability assessment. These self-audits allow organizations to assess their software development processes and assist in formulating an effective program to overcome noted deficiencies. Both LCSECs have training programs focused on deficient areas noted in the assessment and designed to increase their competency levels. The MICOM SED is scheduled to undergo their next self-examination in April of 1994. The LCSEC's realize the importance of training and of constantly improving their process. (Craig, 1994)

F. INTERVIEWS

The following issues were discussed in interviews with senior personnel from both LCSECs.

1. Existing Problems

According to senior management, the biggest problem facing the LCSECs is inconsistent and volatile funding. The LCSECs receive most of their funding as customer reimbursable funding for specific work. If a center loses funding for a project, work on that project is terminated, and the people are transferred to another project or laid off. Generally speaking, a cut in funding translates almost directly to a cut in personnel. Software development and support is a manpower intensive effort. As such, the productivity and quality of the work effort largely depends on the quality and continuity of the work force.

The LCSECs spend a significant amount of their time justifying the work that they perform. The LCSECs leadership is required to continually educate senior resource managers on the effects funding cuts have on PDSS in order to try and maintain their funding levels. They believe they have been easy targets for funding cuts during the initial formulation phase of the budget process. The problems associated with inconsistent and volatile funding have been previously discussed. (Craig, 1994)

Currently, both of the LCSECs have not had problems getting qualified contractor personnel to work on their software projects. However, this is primarily because of the poor national and local economies. Should these economies start to pick up the LCSECs may find it difficult to attract high quality software engineers, especially

given the unstable job security. Both of the LCSECs leadership agreed that funding this year has been tighter than they can ever remember. For the MICOM SED, FY 94 funding came late, forcing it to terminate part of its contractor's work force until funding authority arrived. Its contracts end early in the fiscal year.

Another concern the LCSEC leadership expressed was concerned with how to rebuild a PDSS support structure for a system whose funding has been cut and its PDSS resources dispersed. In FY 94 alone, 26 Army systems under PDSS in the LCSECs had their funding terminated. The CECOM SED designs their support contracts so that they run from 31 November to 1 December. This allows the CECOM SED to archive a program should its support be terminated. Archiving a program involves placing all program documentation and work into a format that will be easier to relearn should funding support resume. For those LCSECs that have contracts that run from 30 September to 1 October, when funding is terminated work just stops and the program cannot be archived unless the LCSEC can come up with funding to complete the archiving. For programs which are not archived, it will be much more difficult to restart the PDSS support. The LCSECs estimate that it would take at least two years to restart the PDSS support for a system and reach the previous level of productivity once its funding has been terminated and its people and resources scattered. (Wagner, 1993)

2. Impact of a Lack of Software Support

A lack of software support creates several challenges for the soldier. From the CECOM SED point of view, a lack of continuous software support degrades the expertise available to perform work on a system. CECOM SED not only fixes software

problems, but sometimes it is the only source of expertise available to the users in the field to fix system-related problems. Many times users in the field call the CECOM SED with system and interoperability problems. Sometimes the reported problems stem from software bugs. But most of the time they simply reflect a lack of system expertise on the part of the users. If the LCSECs are forced to cancel software support for a system, then users may have no one to answer questions related to such interoperability problems. (Turner, 1994)

In addition, a lack of continuous software support raises the system's life cycle costs. It is expensive and time consuming to try to restart a PDSS maintenance effort once that support has been terminated. It is estimated that once support is terminated for a system it takes upwards of two years to restart that support again. (Wagner, 1993)

3. Balancing Budget Cuts in the LCSECs

If the funding for a system is killed, the people involved in support of that system are either moved to a new project or laid off. If the funding for a system is not terminated, but only somewhat reduced, then some of the personnel will be moved to another system or laid off. Any system enhancements will be stretched out. (Turner, 1994)

A second method the LCSECs use to balance budget cuts is to restrict temporary duty, training, and restrict or eliminate cash incentive awards for individuals. During such budget cuts, discretionary research and development money for the centers is also reduced or eliminated. Research and development work helps retain good software engineers in the Government. Many individuals remain because of the challenging work.

Cutting discretionary research and development funding eliminates one of the few methods the LCSECs have to keep high quality individuals in Government service. (Rostollan, 1993)

G. CONCLUSION

The number of systems that the LCSECs are providing support to has skyrocketed. In the MICOM SED the number of systems it supports has increased nearly seven-fold since 1984. In the CECOM SED the number of systems it supports has increased by over 80% since 1988. In the CECOM SED the number of systems that are under development is two and a half times higher than the number currently in PDSS. In the CECOM SED the projected budget requirements are expected to increase four-fold from 1988 to 1995, increasing from $50 million to over $200 million.

The LCSECs are performing all of this work with fewer people. The CECOM SED has been downsized from 360 Government personnel to an eventual strength of 250 individuals. A rather grim picture emerges for the future of PDSS support. The budget decline and personnel cutbacks do not bode well for the support a soldier can expect for a system, given the existing and future software systems requiring support.

VII. CONCLUSIONS AND RECOMMENDATIONS

This research has looked at PDSS activities and the impact of budget and personnel cuts for software support for fielded systems. Conclusions and recommendations from the research are discussed in the following paragraphs.

A. CONCLUSIONS

1. Interoperability

Senior Army leadership should be concerned with the declining budgets and personnel cuts at PDSS centers. Such reductions lead directly to reduction or termination of specific software programs. This can be especially serious and present disastrous consequences for systems which are part of interoperable combat and communications systems.

When AMC and TRADOC performed a FY 1994 OMA funding analysis review of the CECOM SED, the AN/MSC-64 fell below the cut-off line and support was terminated. The AN/MSC-64 is a communications system that the CINCs use to talk to the National Command Authority. It is an item used by all services. The Army provides software maintenance on the system. Because all the services use it, the decision to cancel support for this system soon raised serious questions. Support for the AN/MSC-64 was reinstated.

Cancelling the PDSS support for any piece of equipment can effectively doom that item to early obsolescence. For systems whose functionality requires interoperability

with other systems, this obsolescence may occur even sooner. Changes in other systems or newly introduced systems may render this unsupported equipment unusable if changes to its software cannot be made.

2. Funding Stability

A key factor facilitating the smooth and efficient support of a system is stability of its funding. Funding in PDSS translates directly into personnel resources. As a result, cutting a systems funding usually translates directly to personnel cuts. Because software support is a personnel-intensive activity requiring a significant learning curve and accumulation of program knowledge over time, fluctuations and turnover of personnel are devastating to productivity and quality software support.

Over the past few years the PDSS centers have not had a stable funding base. The amount of funding they receive rarely meets the needed amount. Such funding instability creates an inefficient software maintenance process. At CECOM and MICOM SEDs, over 75% of the PDSS work force is contractor personnel. Unfortunately, the PDSS centers do not have the capability to make up the personnel shortfall by using in-house programmers. Contractors are continually having to retrain personnel who have been laid off or transferred because of funding instabilities. Tremendous inefficiencies are created, resulting in increasing software backlogs. Army leadership needs to rectify these funding fluctuations. Because they directly effect people, they have a much more serious affect than budget cuts that primarily impact day-to-day operations, training, or travel. The Army is currently not facing up to the true cost of supporting its software by the way it currently under-budgets for software support and later adds additional funding

when the CINCs express the importance of continued support. Tremendous productivity is lost and the ability to effectively plan practically vanishes.

3. Growth in Systems Supported

It can be concluded from the data presented that the number of systems to be supported will increase dramatically over the next several years. If current budget and personnel reductions continue, the Army will find itself unable to provide the necessary PDSS for the ever-growing number of fielded systems.

The fact that the number of sites or quantity of systems fielded are significantly reduced because of the Army downsizing has very little impact on the funding and personnel required to provide an adequate amount of support for a system. For instance, whether a communication system has 5000 units or 1000 has little impact on the number of people required to support the software and firmware. A similar, situation arises in aviation systems. Whether we operate 2000 aircraft or 500, the resources (personnel and funding) required to provide adequate support does not differ by much.

Another factor that deserves consideration is that the number of systems (and the amount of associated software) under development far outnumber the systems currently receiving PDSS. These new systems tend to be much more software intensive and complex. Now is the time to review the life cycle support resources necessary to ensure adequate support for these new systems when they are fielded.

4. Reductions in PDSS Center Personnel

The current environment of Government downsizing is having a detrimental effect on the PDSS centers. The PDSS centers face personnel reductions from two perspectives. The number of billet assignments is being reduced and a hiring freeze prevents hiring new people to fill existing vacancies. The work these PDSS centers performs does not go away. There has not been a reduction in work. Rather, the number of new systems and the quantity of software is increasing. In general it can be said PDSS work is largely contracted out to the civilian sector. It is not possible for the Government to perform all of the PDSS work given the number of Government employees. To do so requires a trained cadre of software professionals, many of whom currently serve as Government oversight personnel.

5. The PDSS Center Grade Structure

The grade structure of the PDSS centers, combined with other factors, has the potential to create severe personnel turn-over problems in the PDSS centers. The current PDSS grade structure offers programmers very little promotion opportunity. Most jobs are in grades GS 12 and 13. If an individual wants to get promoted, he or she is forced to leave the PDSS center. While at present relatively poor local and national economies have kept people from leaving their jobs and moving, once the economy starts to pick up these shortsighted management methods could cost PDSS centers many of their trained software professionals. Unfortunately, it is often the ones that are best trained and most knowledgeable that leave because they are more marketable and in demand.

6. Contractor Support of PDSS Centers

While the ratio of contractor-to-Government personnel varies somewhat between centers, the trend is for a greater proportion of contractors doing the support at the Army PDSS centers. What level of contractor support will be allowed in the PDSS centers? Attention should be focused on this issue and a determination made. If the present trend of high contractor-to-Government ratio continues, the Army risks changing PDSS centers into contract management agencies where the Government employees perform little actual software engineering. The Government's ability to attract quality software professionals (who want to do software work) will seriously diminish under such conditions.

B. RECOMMENDATIONS

1. PDSS Funding Support

A systems software support should be treated in the same manner as the systems post-deployment logistics support. At present, there seems to be a belief that software support can be significantly reduced or even terminated without much affect on the soldier's ability to use a system.

Commanders would not think of continuing to operate equipment for which all logistics support had been terminated. The same concept applies for a system for which software support has been seriously degraded. If we can no longer support a system's software, we should seriously consider retiring the system from the field. To do otherwise requires the soldier to use a substandard piece of equipment. If a decision is

made to terminate a system's software support funding, Army leadership must also give thought to removing that system from the force structure.

2. PDSS Funding Stability

Every effort must be made by people involved in the PPBS process to ensure that a PDSS center's funding reflects its workload. Without a consistent funding profile, the Army will see ever-increasing shortfalls in a system's performance. This up-and-down thrashing and the insecurity that results means decreasing productivity and quality.

3. PDSS OMA Funding

The OMA funding system for software is inadequate. One alternative is for program managers or the post-deployment material managers to be given responsibility for managing and tracking PDSS funds for their systems. These funds could then be tracked as part of total system funding, rather than individual accounts that can be targeted for cuts.

4. PDSS Personnel Cuts

PDSS centers need to be exempted from the current and future rounds of personnel RIFs and downsizing presently taking place. Both of the PDSS centers surveyed already exceed a contractor-to-Government ratio of 75/25. Further downsizing will increase the ratio of contractor personnel. This will only lead to a decrease in the Government's ability to properly support its software. The Army should study this issue and determine what the maximum contractor-to-Government ratio is going to be. Once

that level is determined, the personnel strengths and authorizations must be increased to reflect those levels and the present hiring freeze lifted.

5. Discretionary Funding

The Army needs to increase the discretionary research and development funding available to the PDSS centers to insure that quality personnel will stay with the PDSS centers once the economy starts to pick up. Incentives and/or promotions should also be examined.

This draws to a close this discussion of challenges and shortfalls in Army software support. As we approach the next century, the Army will have significantly greater amounts of software to support. The programs are larger and more complex. They will require a much bigger support effort and significant amounts of funding above today's systems. Army leadership should take the lead on resolving the many issues raised in this discussion and plan for the future challenges software-intensive systems will present.

APPENDIX

A. PEOPLE:

1. The PDSS Work force

    a. Demographics of the work force
        Government   Civilian   Military

    b. What is the rank structure of the work force?

    c. Is a hiring freeze in place?

    d. What is your turnover rate?

    e. If a hiring freeze is in place how do you handle the turnover rate?

    f. How long does the average person stay in your organization?

    g. Do you have problems recruiting people to work in your organization?

2. What is the History of the work force since 1985?

3. What is projected to happen to the work force?

4. Can you give me a worst case problem that has to do with people and your cuts in supporting the systems.

B. TRAINING:

1. What are the education levels of people in your activities?

2. What skills do you have in your activity?

3. What type of training do you have in your activity?

4. How often do you perform your training?

5. Have you undergone a software capability assessment as given by the Software Engineering Institute, Carnegie Mellon University?

6. If you have, what was your competency level?

7. Do you have assessment teams to perform this function in support of program manager contractor evaluations for contract awards.

C. BUDGET:

1. What has the history of the budget from 1985 been?

2. What is projected to happen in the future to the budget?

3. What are the five most serious impacts of the declining budget?

4. What are the implications of budget cuts? How do you handle or notify the contractor that funding levels have been cut?

5. What are your feelings on the budget cuts? Are you being cut more or less than everybody else?

6. How big a problem is the budget cut, or is it a problem?

7. What are the worst case problems of budget problems in supporting your systems?

D. OPERATIONS:

1. What software have you supported since 1985?

2. How many languages have you supported since 1985?

3. How many systems have you supported since 1985?

4. How many programs have you supported since 1985?

5. How many Ada programs have you supported since 1985?

6. What is the role of the contractor in your activity?

7. How do you classify and prioritze your work?

8. How do you balance the workload?

9. How do you handle System crashes? Describe your scheme to fix them.

10. How do you handle safety issues? Describe your scheme to fix them.

11. What is the backlog of maintenance?

12. Is the backlog of maintenance increasing?

13. Do you have an idea of what your backlog has been since 1985?

14. What kind of support equipment do you have?

15. How do you track your percentage of memory used, and how do you communicate requirements for hardware upgrades?

16. What was your role during Operation Desert Shield/Desert Storm?

E. LEADERSHIP INTERVIEWS:

1. What is the impact of existing problems, i.e. what do you do when you run out of money?

2. What is the Impact of a lack of software support?

3. How do you balance the cuts; people, equipment, support, TDY?

4. What is coming down the line in your area?

5. What are your top five concerns right now?

LIST OF REFERENCES

1. U.S. Air Force, *Air Force Acquisition Model (AFAM)*, version 1.4, ASC/CYM, Wright Patterson Air Force Base, Ohio, 45433-6503, 1993

2. *Army Executives For Software (ARES)*, CECOM SED, 12 July 1991, p. 9-21.

3. Boehm, B., "Software and Its Impact: A Quantitative Assessment," *Datamation*, Vol. 19, No. 5, May 1973

3. Brooks, F., *The Mythical Man-Month*, Addison-Wesley Publishing Co., Inc., 1975

4. Buckley, W., *Computer Makers Feel Key to Sales Lies in Better Programming*, Wall Street Journal, Sept 29, 1980, p. 1,18.

5. CECOM SED, *Briefing to Department of Defense IG*, Briefing to DoD IG, 15 November 1993

6. CECOM SED, *Golden Nugget Vugraphs*, February 1994.

7. CECOM SED, *LIFE CYCLE SOFTWARE ENGINEERING FOR MISSION CRITICAL DEFENSE SYSTEMS*, Briefed by John H. Sintic Director CECOM RDEC SED, 19 Mar 1993

8. CECOM SED, *RDEC PROGRAM REVIEW SED OVER VIEW*, Briefer John H. Sintic, 21 October 1993

9. CECOM SED, *SOFTWARE "MAINTENANCE": SOME ARMY LESSONS LEARNED*, Briefer Dennis Turner, 21 February 1994

10. Interview between B. Craig, Director MICOM SED, Redstone Arsenal, Al., and the author, 30 November 1993 and 24 February 1994.

11. Department of Defense, *Department of Defense Instruction Number 5000.2, Defense Acquisition Management Policies and Procedures*, Department of Defense, Government Printing Office, Washington, DC, February 1991, p. 6-B-2.

12. Department of Defense, *Department of Defense Standard (DOD-STD) 2167A, Defense System Software Development*, Department of Defense, Government Printing Office, Washington, DC, 29 February 1988

13. Interview between Joe Doughtery, Chief Plans and Programs, CECOM SED, and the author, 17 December 1993.

14. Freedman, D., and Weinberg, G., "Maintenance Reviews", *Techniques of Program and System Maintenance*, Lincoln, NE: Ethnotech, 1980.

15. General Accounting Office (GAO), *GAO/IMTEC-92-62BR Defense's Embedded Software Costs*, Government Printing Office, Washington, DC, 6 July 1992.

16. Gibson, V., *A Study of Complexity Metrics as Surrogate Measures of Software Maintainability*, Ph.D. Dissertation, State University of New York at Binghamton, Binghamton, New York, 1986

17. Lientz, B., and Swanson, E., *Software Maintenance Management A Study of the Maintenance of Computer Application Software in 487 Data Processing Organizations*, Addison-Wesley Publishing Company, 1980.

18. Martin, James, and McClure, C., *Software Maintenance - The Problem and its Solutions*, Englewood Cliffs, NJ: Prentice Hill Inc., 1983.

19. Defense Systems Management College, *Mission Critical Computer Resources Management Guide MCCR*, Government Printing Office, Washington, DC, January 1990

20. Mills, H., "Software Development", *IEEE Trans. on Software Engineering*, Vol. SE-2, no. 4 December 1976: p. 265-273.

21. Osborne, W., and Martin, R., *Computer Science and Technology Guidance on Software Maintenance*, U.S. National Bureau of Standards, NBS Special Publication 500-106, December 1983

22. Pizzarello, A., *Development and Maintenance of Large Software Systems*, Lifetime Learning Publications, 1984

23. Interview between Nancy Rostollan, Chief Business Management Office, MICOM SED, and the author, 16 December 1993

24. MICOM SED, Unclassified, Rostollan, Nancy, Chief Business Management Office, MICOM SED, Memorandum, 22 December 1993.

25. Schneiderman, B., *Software Psychology*, Cambridge, MA: Winthrop Publishers, Inc., 1980.

26. Swanson. E., "The Dimensions of Maintenance", *2nd International Conference on Software Engineering, Proceedings San Francisco*, October 13-15, 1976, P. 492-497

27. Interview between Dennis Turner, Deputy Director, CECOM SED, and the author, 22 February 1994

28. Interview between Jim Wagner, Associate Director Life cycle Software Support, CECOM SED, and the author, 30 November 1993 and 22 February 1994

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, California 93943-5100	2
3.	Professor David V. Lamm (Code SM/LT) Naval Postgraduate School Monterey, California 93943-5100	2
4.	Professor Martin J. McCaffrey (Code AS/MF) Naval Postgraduate School Monterey, California 93943-5100	6
5.	Professor James C. Emery (Code AS/EY) Naval Postgraduate School Monterey, California 93943-5100	1
6.	Director CECOM SED Attn: Mr. Dennis J. Turner U.S. Army CECOM, Ft Monmouth, NJ 07703-5207	2
7.	Director MICOM SED Attn: Mr. Bill Craig U.S. Army RDEC Redstone Arsenal, Al 35898-5260	2
8.	Cpt Mark C. Jones 2205 Viewmount Springfield, Or 97477	1